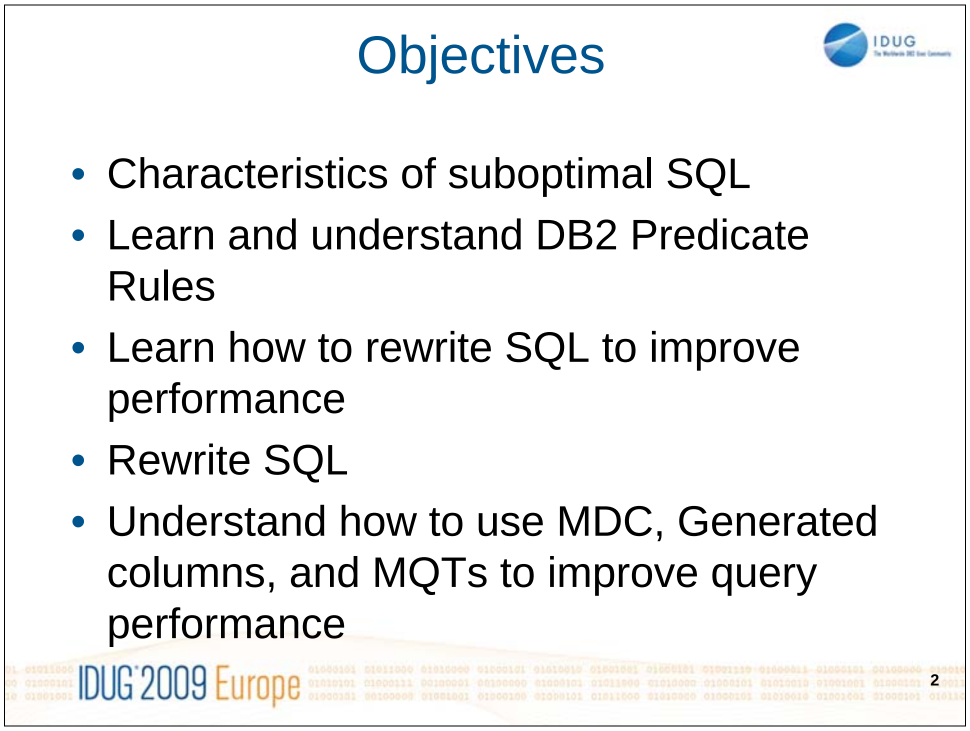
Session: C11

# SQL Tuning for Toughies, Top Tips, Techniques and Solutions

Phil Gunning
Gunning Technology Solutions, LLC

**7 October 2009 • 11:00 – 12:00am**
**Platform: DB2 LUW**

IDUG
The Worldwide DB2 User Community

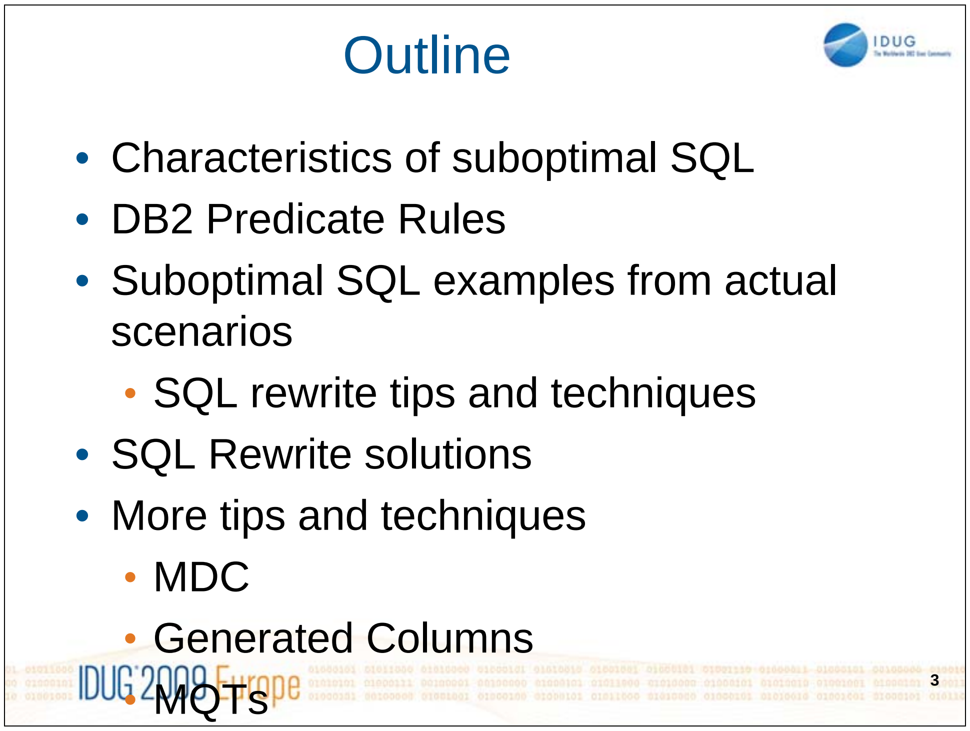


## Objectives

- Characteristics of suboptimal SQL
- Learn and understand DB2 Predicate Rules
- Learn how to rewrite SQL to improve performance
- Rewrite SQL
- Understand how to use MDC, Generated columns, and MQTs to improve query performance

IDUG 2009 Europe

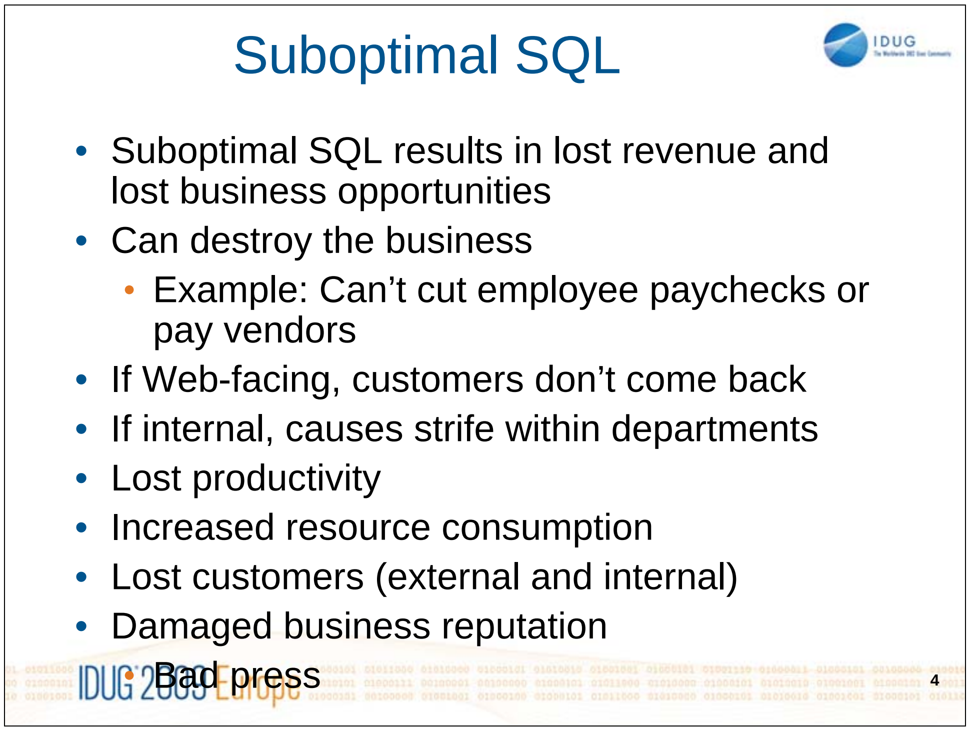
NOTES: The goal of this presentation is to provide you with some examples and tips and techniques for rewriting SQL to improve performance. In the past, DBAs spent a lot of time on database configuration issues but with big improvements in autonomic configuration settings over the last several releases of DB2, DBAs now don't have to spend as much time on configuration issues, so more time can be spent helping developers identity and tune suboptimal SQL. Suboptimal SQL is the #1 problem facing developers and DBAs alike.



## Outline

- Characteristics of suboptimal SQL
- DB2 Predicate Rules
- Suboptimal SQL examples from actual scenarios
  - SQL rewrite tips and techniques
- SQL Rewrite solutions
- More tips and techniques
  - MDC
  - Generated Columns
  - MQTs

NOTES:

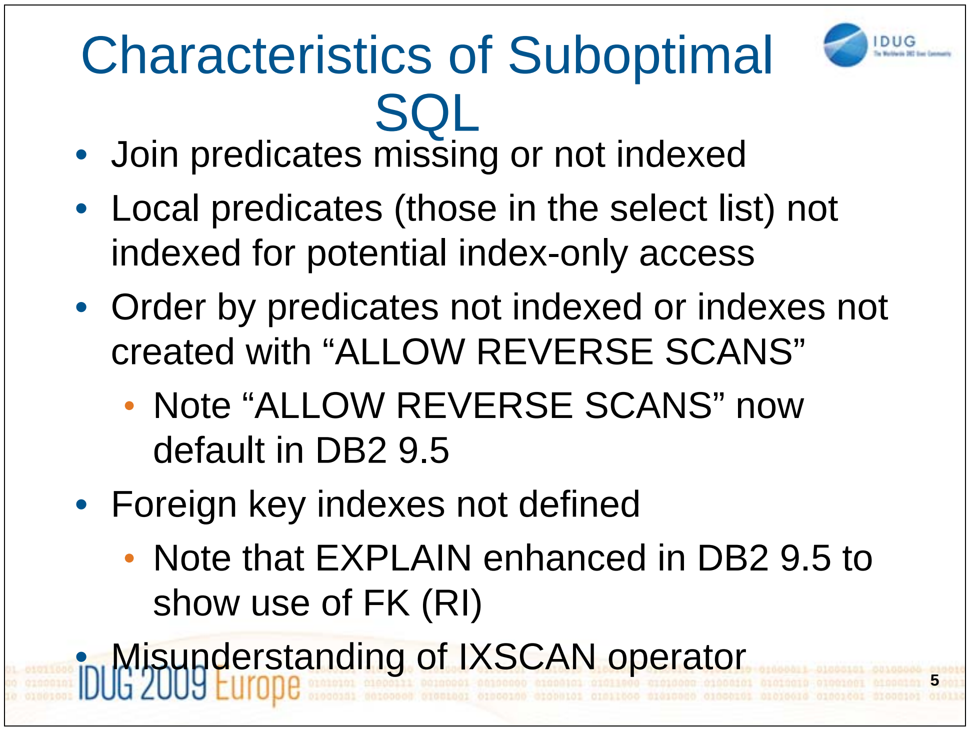


## Suboptimal SQL

- Suboptimal SQL results in lost revenue and lost business opportunities
- Can destroy the business
  - Example: Can’t cut employee paychecks or pay vendors
- If Web-facing, customers don’t come back
- If internal, causes strife within departments
- Lost productivity
- Increased resource consumption
- Lost customers (external and internal)
- Damaged business reputation
  - Bad press

IDUG 2009 Europe 4

NOTES: Suboptimal SQL costs companies millions, possibly billions of dollars a year on lost revenue due to the causes listed above.

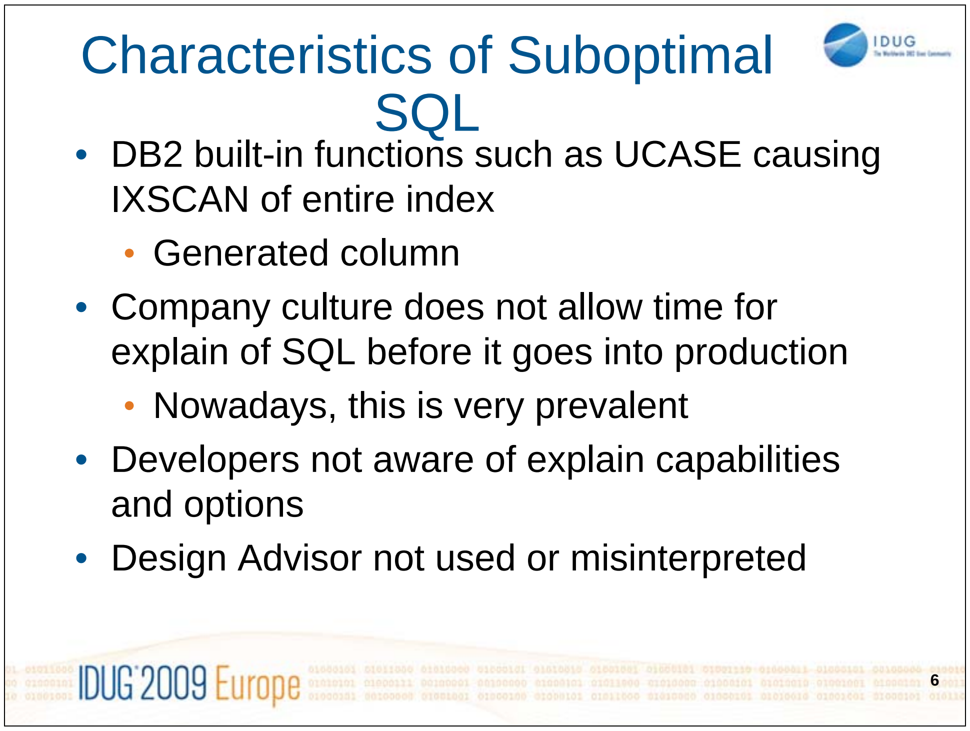


## Characteristics of Suboptimal SQL

- Join predicates missing or not indexed
- Local predicates (those in the select list) not indexed for potential index-only access
- Order by predicates not indexed or indexes not created with “ALLOW REVERSE SCANS”
  - Note “ALLOW REVERSE SCANS” now default in DB2 9.5
- Foreign key indexes not defined
  - Note that EXPLAIN enhanced in DB2 9.5 to show use of FK (RI)
- Misunderstanding of IXSCAN operator

IDUG 2009 Europe 5

NOTES: Most of the time with suboptimal SQL I find that indexes have not been created on the correct columns, such as those listed above. And in prior releases of DB2, most third party vendor packages did not create indexes with “Allow Reverse Scans” specified and most shops are reluctant to change the provided DDL.

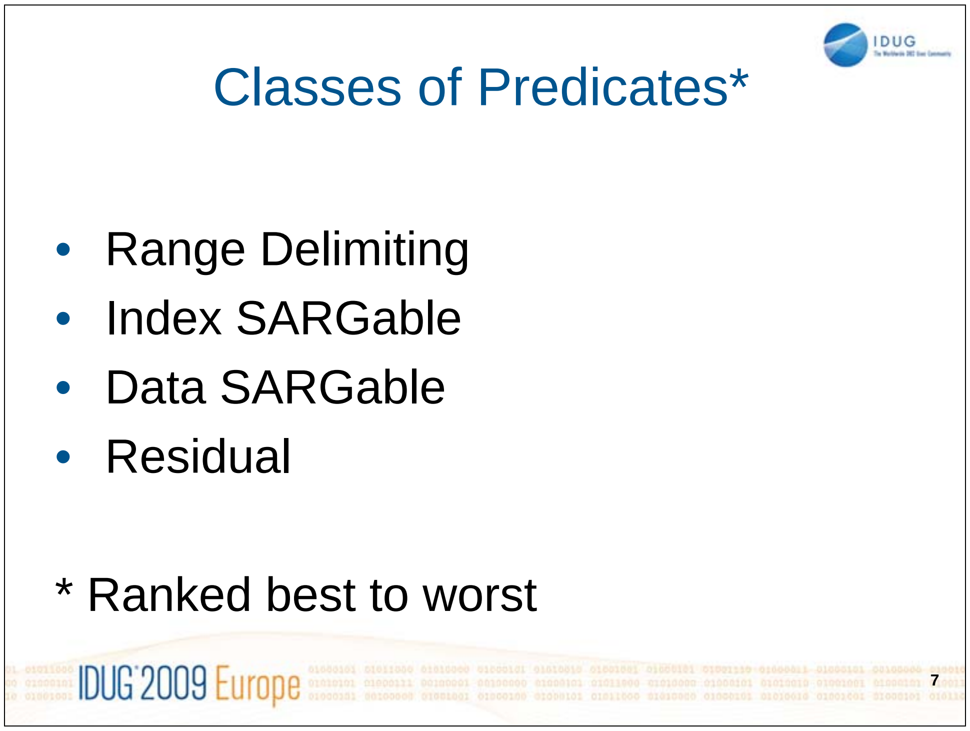


## Characteristics of Suboptimal SQL

- DB2 built-in functions such as UCASE causing IXSCAN of entire index
  - Generated column
- Company culture does not allow time for explain of SQL before it goes into production
  - Nowadays, this is very prevalent
- Developers not aware of explain capabilities and options
- Design Advisor not used or misinterpreted

IDUG 2009 Europe 6

NOTES: This slide contains some additional characteristics of suboptimal SQL. A big problem today is that most company cultures do not generally allow time for SQL to be analyzed and explained prior to going to production. Most times a QA function does not capture poor performing SQL prior to it going to production.



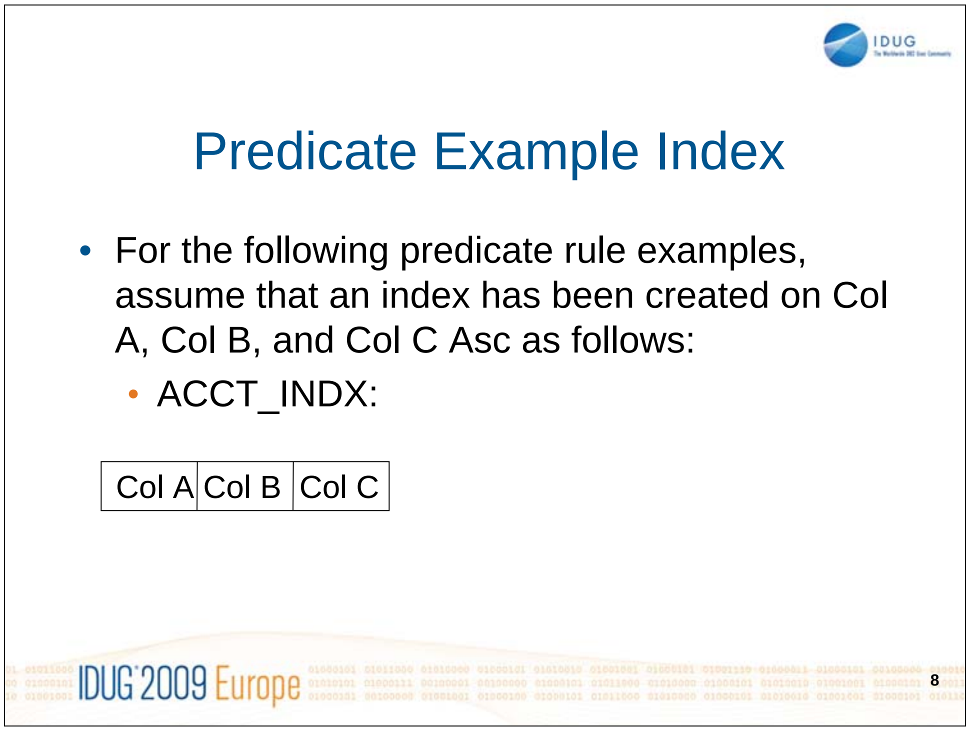
## Classes of Predicates*

- Range Delimiting
- Index SARGable
- Data SARGable
- Residual

* Ranked best to worst

IDUG 2009 Europe 7

NOTES: In order to write good performing SQL, you need to understand the 4 classes of predicates and understand how they are used.



## Predicate Example Index

- For the following predicate rule examples, assume that an index has been created on Col A, Col B, and Col C Asc as follows:
  - ACCT_INDX:

| Col A | Col B | Col C |
|---|---|---|

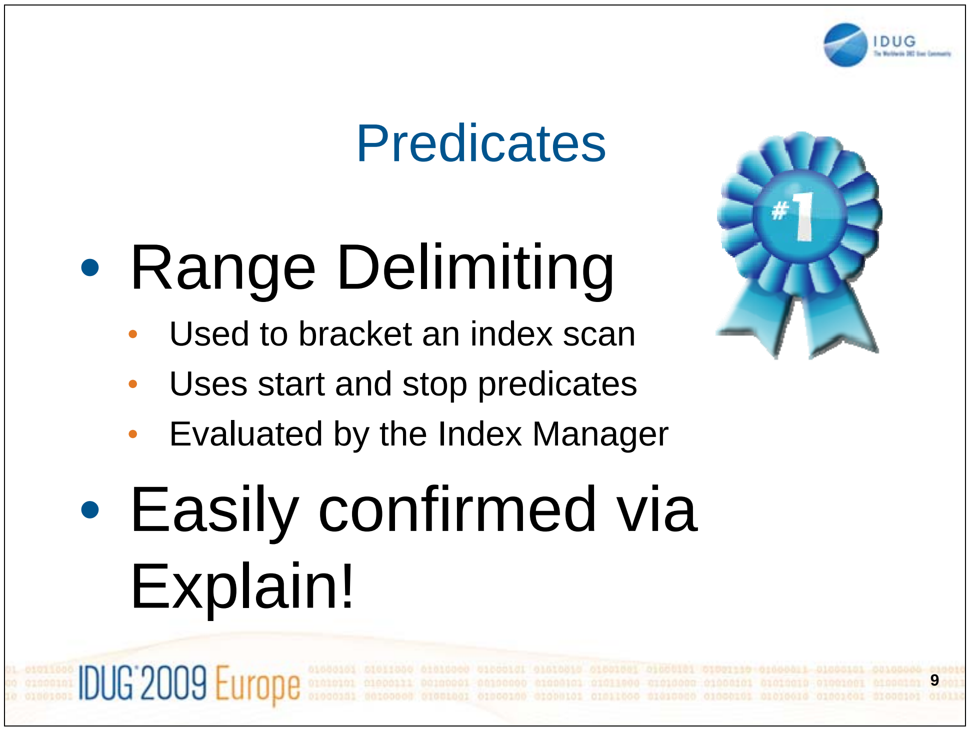
IDUG 2009 Europe 8

NOTES: This slide describes the columns and index created for the predicate rule examples on subsequent slides.  The ACCT_INDX has been created on the following columns.

COL A

COL B

COL C

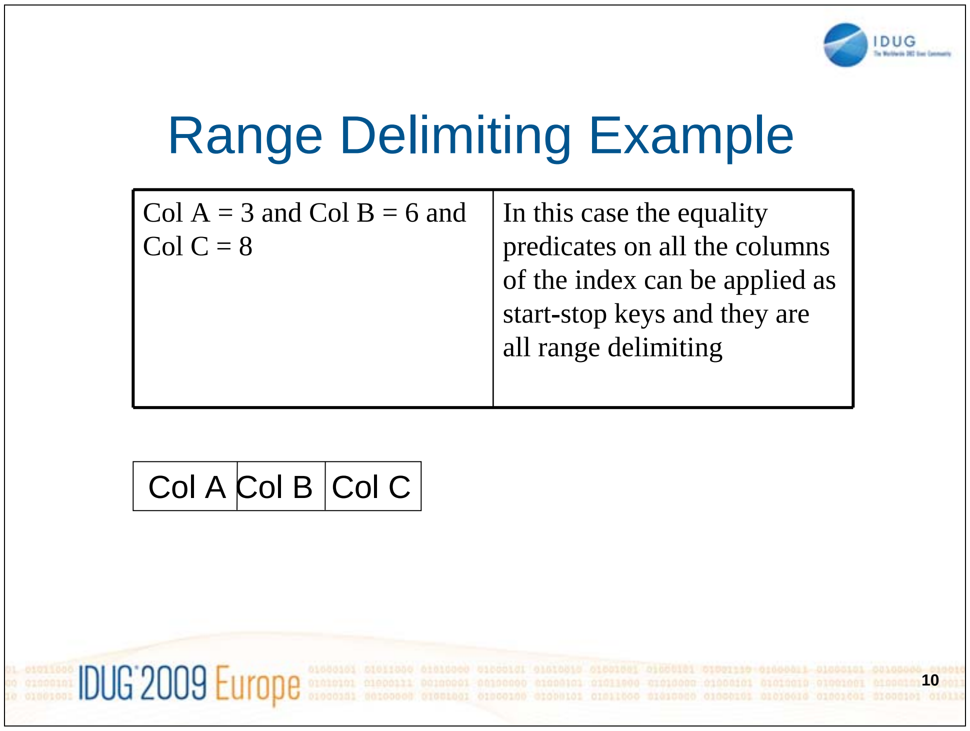


## Predicates

- Range Delimiting
  - Used to bracket an index scan
  - Uses start and stop predicates
  - Evaluated by the Index Manager
- Easily confirmed via Explain!

IDUG 2009 Europe 9

NOTES: Range delimiting predicates are used as start, stop or start-stop predicates for index access. Start-stop predicates are similar to matching index access on DB2 for z/OS. You will see a simple explanation on the next slide and there will be an example and solution discussed later in the presentation.



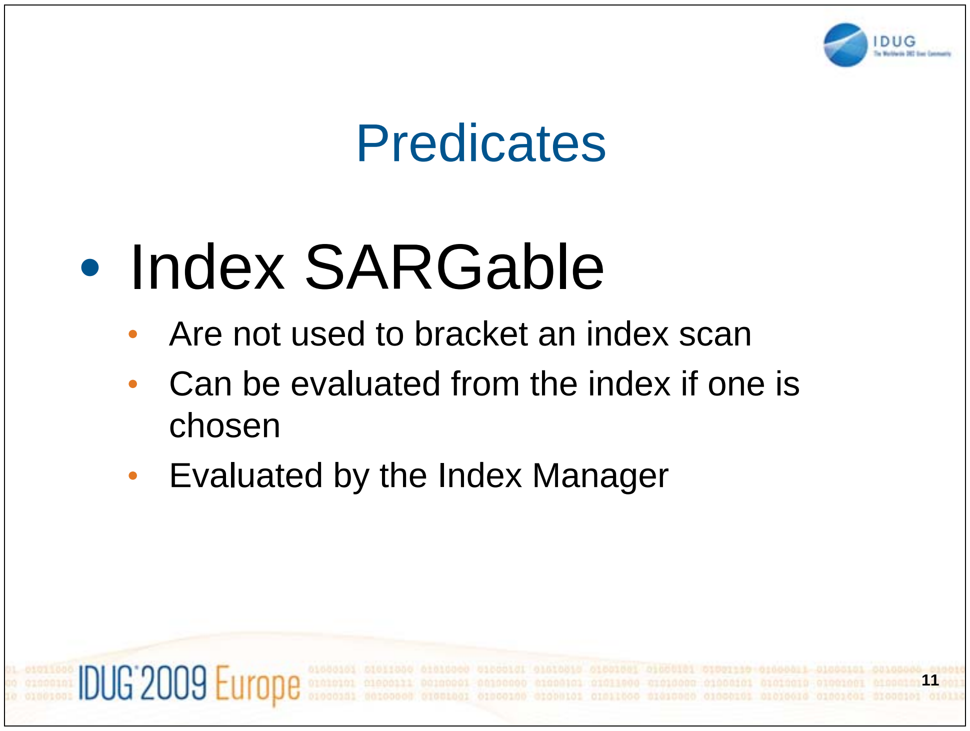
## Range Delimiting Example

| | |
|---|---|
| Col A = 3 and Col B = 6 and Col C = 8 | In this case the equality predicates on all the columns of the index can be applied as start-stop keys and they are all range delimiting |

| | | |
|---|---|---|
| Col A | Col B | Col C |

IDUG 2009 Europe 10

NOTES: Range delimiting predicates are used to bracket an index scan. They provide start and stop key values for an index search or match (start-stop). And they are evaluated by the Index Manager.

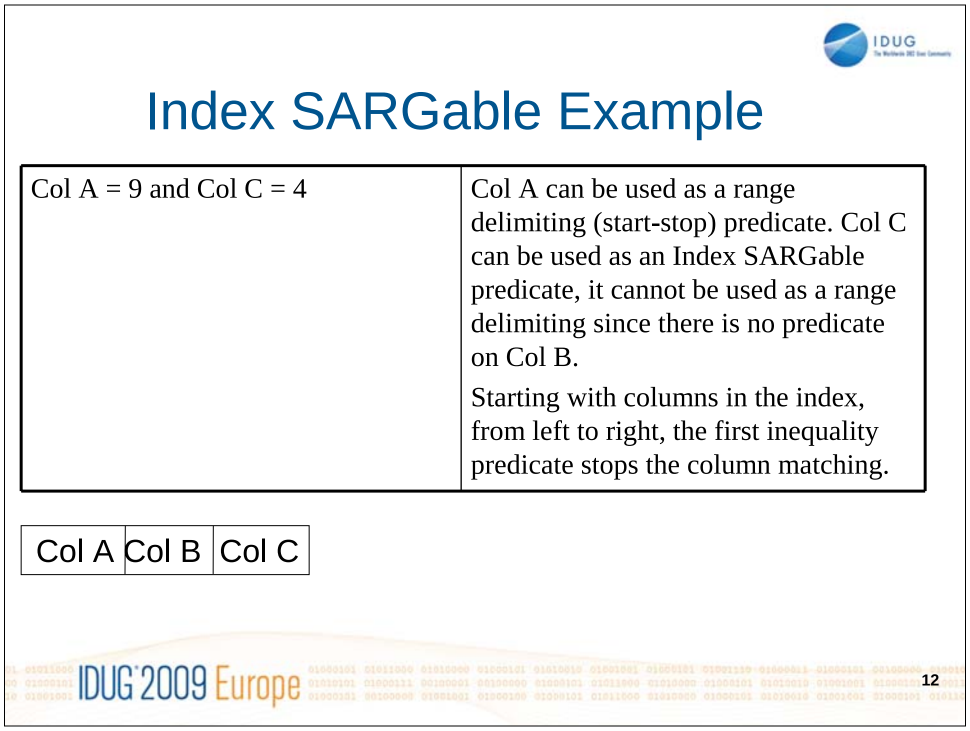


## Predicates

- Index SARGable
  - Are not used to bracket an index scan
  - Can be evaluated from the index if one is chosen
  - Evaluated by the Index Manager

IDUG 2009 Europe 11

NOTES: Index SARGable predicates can be evaluated from the index if one is defined. Index SARGable predicates can be used in conjunction with range-delimiting predicates to provide index-only access. They also provide for Index Scans where there is no matching column but other columns in the index can be evaluated. Index scans can provide good performance for tables that are well indexed, and when dealing with large tables. Since index entries are usually much smaller than data rows, scans usually go against only a subset of the data. Also, indexes tend to be better cached in the buffer pool versus tables.  However, they are not as good as range-delimiting and you should use range delimiting whenever possible.



## Index SARGable Example

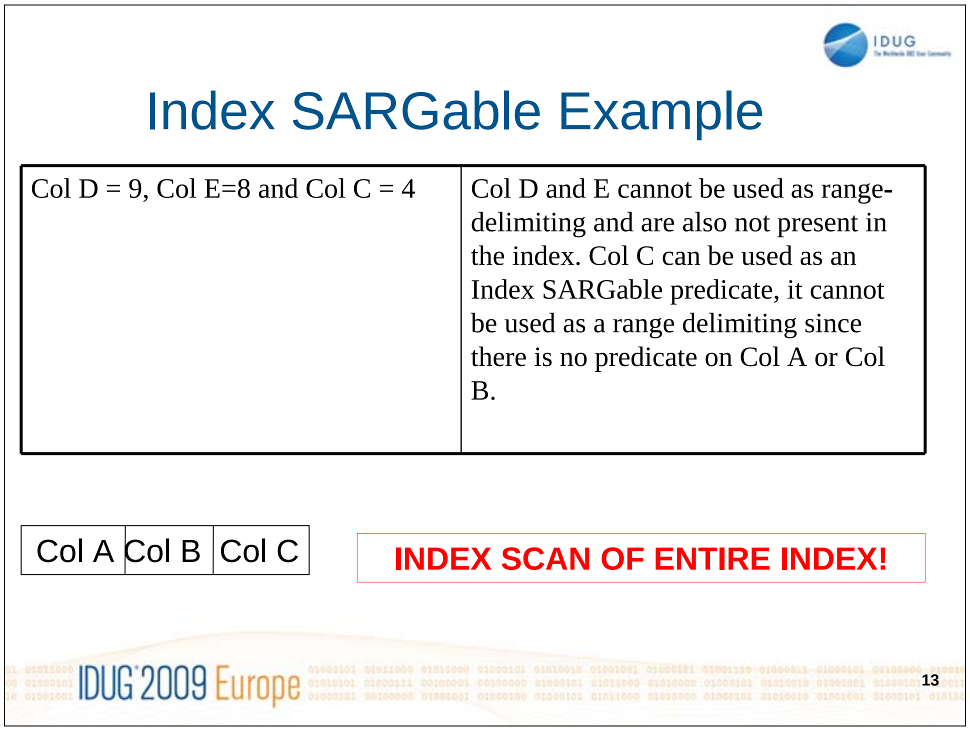
| | |
|---|---|
| Col A = 9 and Col C = 4 | Col A can be used as a range delimiting (start-stop) predicate. Col C can be used as an Index SARGable predicate, it cannot be used as a range delimiting since there is no predicate on Col B.<br>Starting with columns in the index, from left to right, the first inequality predicate stops the column matching. |

| Col A | Col B | Col C |
|---|---|---|

IDUG 2009 Europe

12

NOTES: As previously noted, in all of the predicate examples we are working with composite index that has been created on COLA, COLB, COLC in ASC order.  In this case since Col B is not specified, Col A can be used as a start predicate identify the rows that satisfy the equality predicate, and Col C can be retrieved from the index, however it cannot be used as range-delimiting



## Index SARGable Example

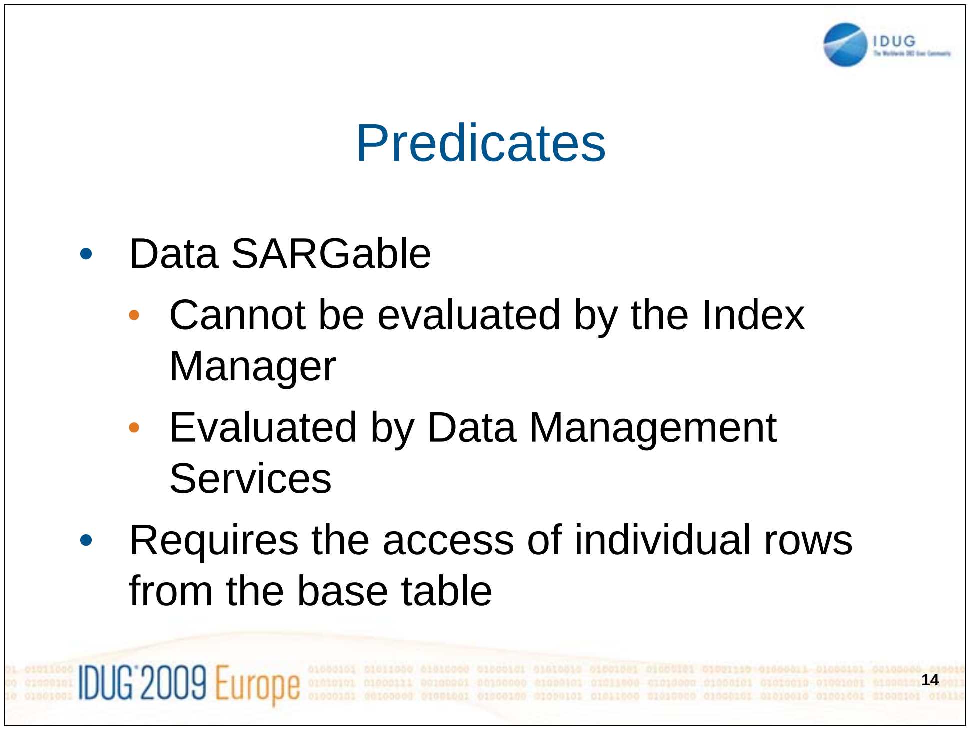
| | |
|---|---|
| Col D = 9, Col E=8 and Col C = 4 | Col D and E cannot be used as range-delimiting and are also not present in the index. Col C can be used as an Index SARGable predicate, it cannot be used as a range delimiting since there is no predicate on Col A or Col B. |

| Col A | Col B | Col C |
|---|---|---|

**INDEX SCAN OF ENTIRE INDEX!**

IDUG 2009 Europe 13

NOTES: As previously noted, in all of the predicate examples we are working with composite index that has been created on COLA, COLB, COLC in ASC order.  In this case, since neither Col A or Col B are specified, range-delimiting predicates or matching predicates is not possible. Col C is present in the index and the optimizer may choose to use it to identify the qualifying rows. However, the entire index will be scanned per row. You will see how to tell the type of predicate by using explain.

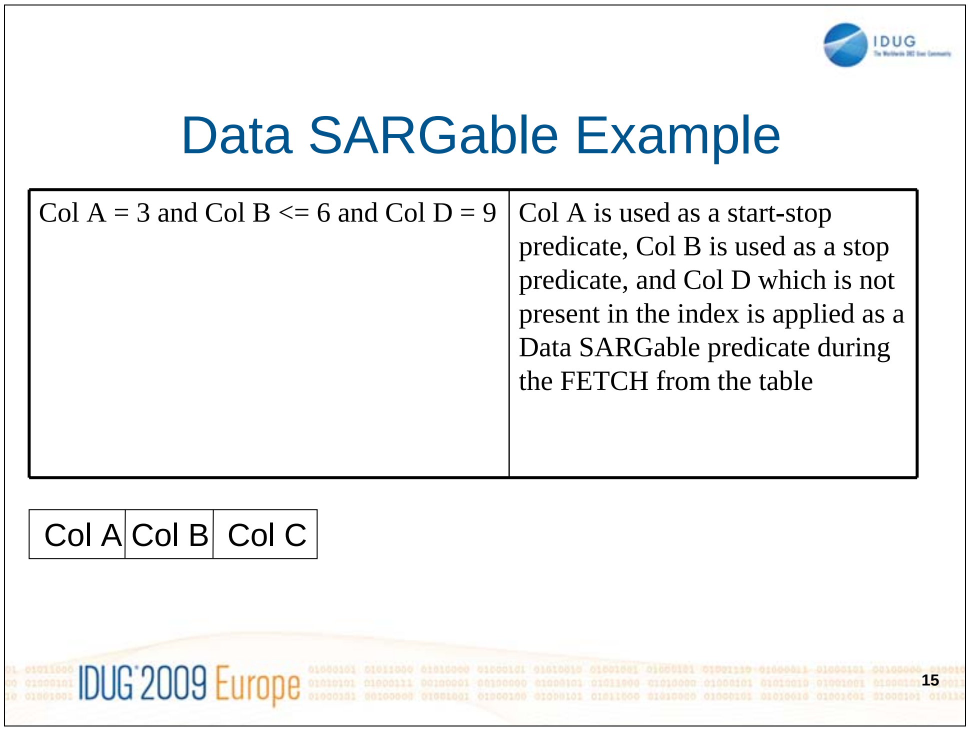


## Predicates

- Data SARGable
  - Cannot be evaluated by the Index Manager
  - Evaluated by Data Management Services
- Requires the access of individual rows from the base table

IDUG 2009 Europe 14

NOTES: Data SARGable predicates are not indexable and must be applied against the base table.



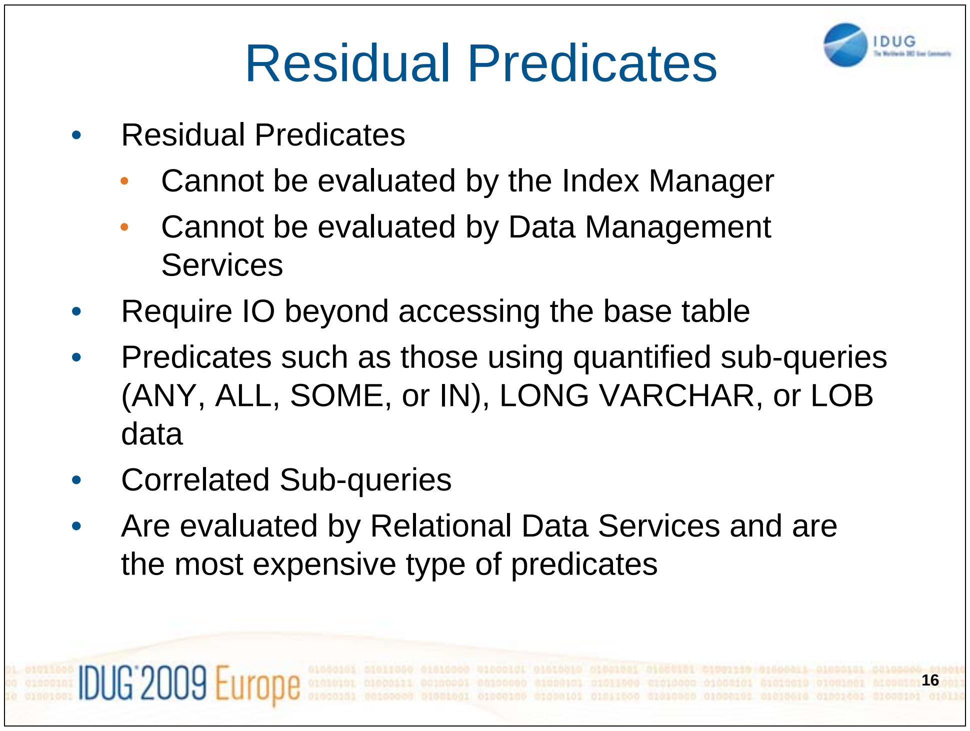
## Data SARGable Example

| | |
|---|---|
| Col A = 3 and Col B <= 6 and Col D = 9 | Col A is used as a start-stop predicate, Col B is used as a stop predicate, and Col D which is not present in the index is applied as a Data SARGable predicate during the FETCH from the table |

| Col A | Col B | Col C |
|---|---|---|

IDUG 2009 Europe 15

NOTES: Data SARGable (SARGable stands for predicates that can be used as Search arguments) predicates cannot be evaluated by the Index Manager. Are evaluated by Data Management Services and require the access of individual rows from the base table.

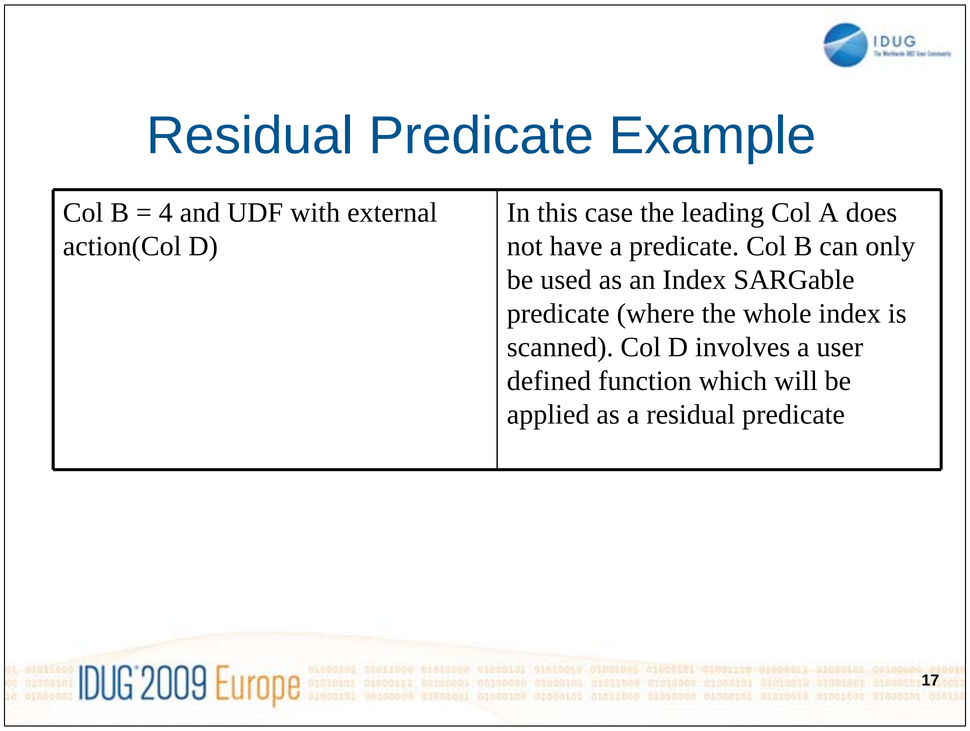


## Residual Predicates

- Residual Predicates
  - Cannot be evaluated by the Index Manager
  - Cannot be evaluated by Data Management Services
- Require IO beyond accessing the base table
- Predicates such as those using quantified sub-queries (ANY, ALL, SOME, or IN), LONG VARCHAR, or LOB data
- Correlated Sub-queries
- Are evaluated by Relational Data Services and are the most expensive type of predicates

IDUG 2009 Europe 16

NOTES: Residual predicates are the worst performing and should be targeted for improvement whenever possible. Residual predicates are evaluated by RDS at a higher level of the engine just before data being returned to the application. Correlated subqueries are in this category.



## Residual Predicate Example

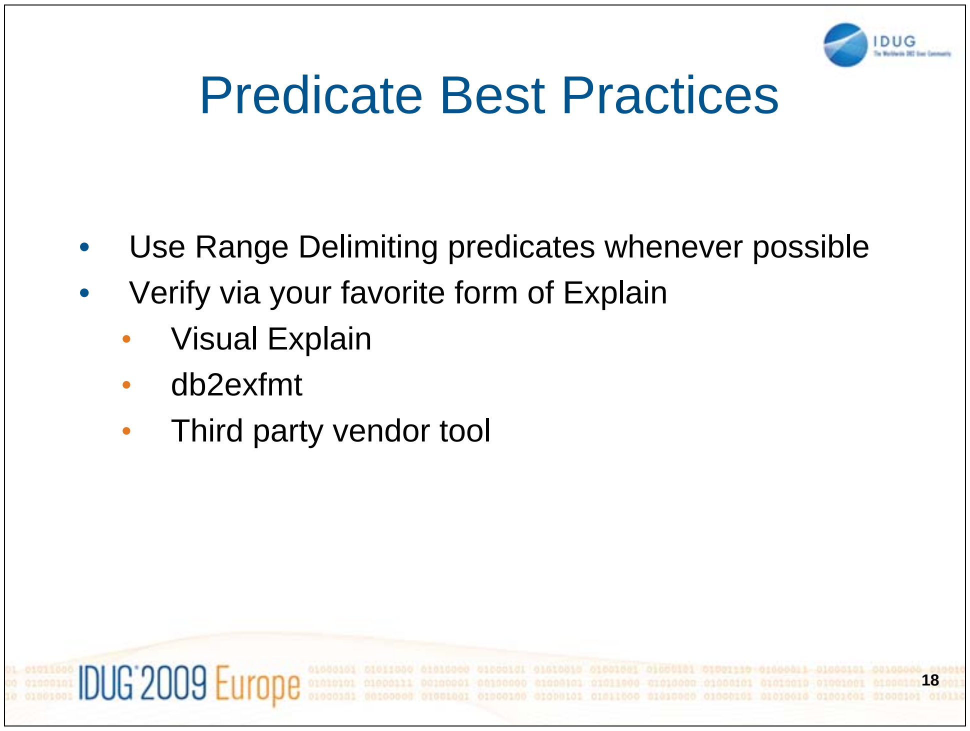
| | |
|---|---|
| Col B = 4 and UDF with external action(Col D) | In this case the leading Col A does not have a predicate. Col B can only be used as an Index SARGable predicate (where the whole index is scanned). Col D involves a user defined function which will be applied as a residual predicate |

NOTES: Residual predicates are applied by Relational Data Services and at the highest level in the engine. Residual predicates should be avoided in high volume OLTP applications where response time is key.  Again, think of residual as “what’s leftover after most processing”. If you follow rules for UDFs without external action they can be indexable as follows:

• predicate specification is present in the CREATE FUNCTION statement

• the UDF is invoked in a WHERE clause being compared (syntactically) in the same way as specified in the predicate specification

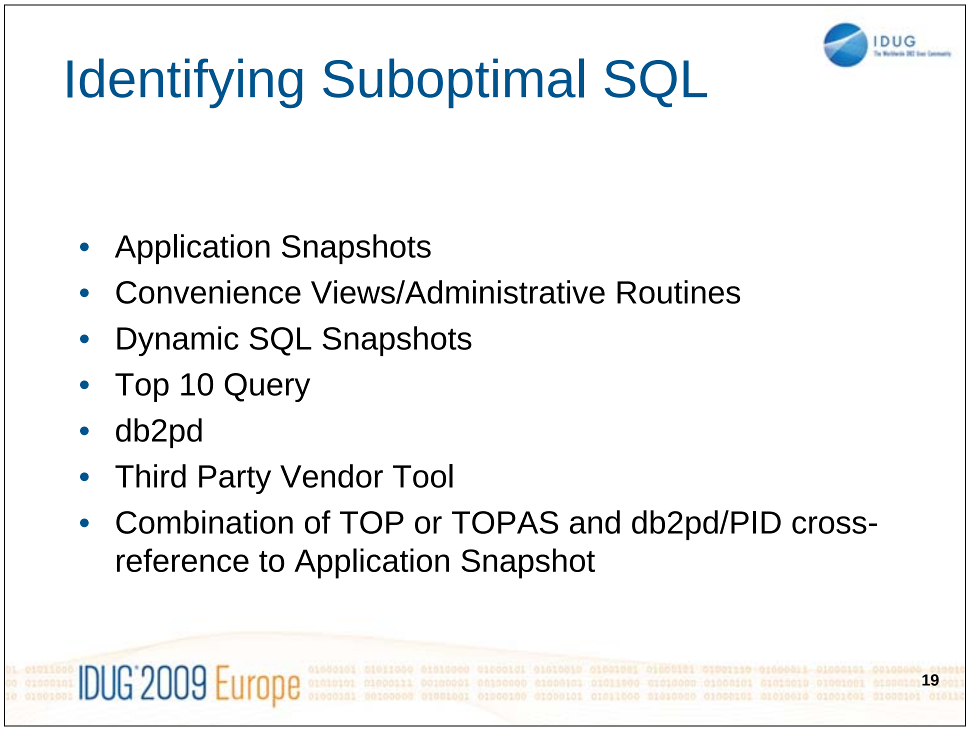
• there is no negation (NOT operator)



## Predicate Best Practices

- Use Range Delimiting predicates whenever possible
- Verify via your favorite form of Explain
  - Visual Explain
  - db2exfmt
  - Third party vendor tool

NOTES: It’s worth repeating.  Use range delimiting predicates whenever possible, especially in transactional based applications.

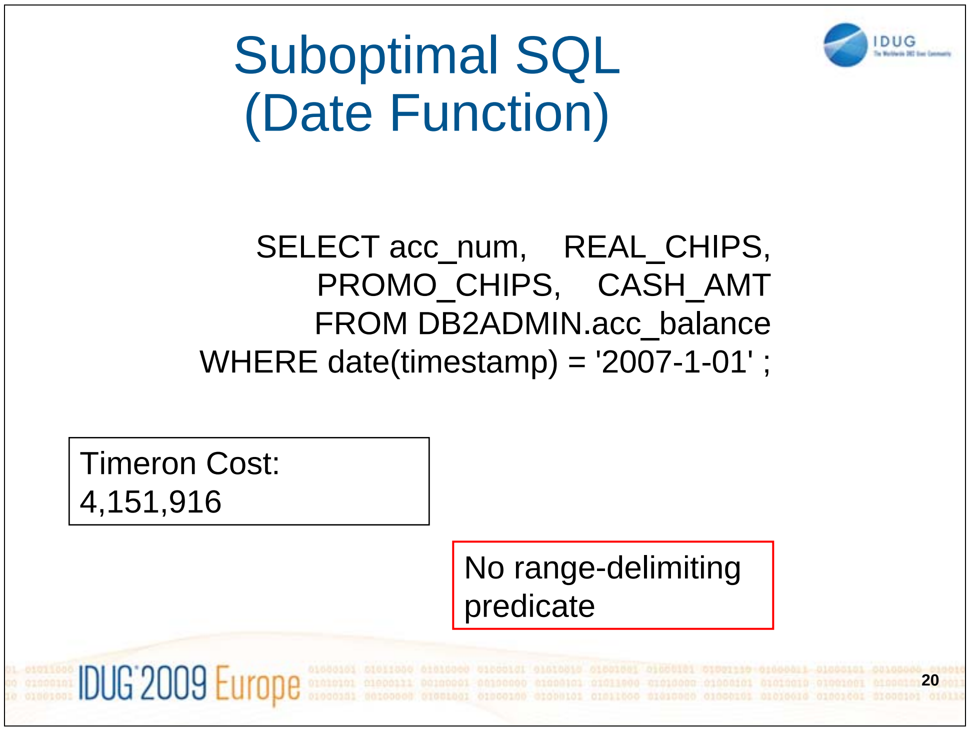


## Identifying Suboptimal SQL

- Application Snapshots
- Convenience Views/Administrative Routines
- Dynamic SQL Snapshots
- Top 10 Query
- db2pd
- Third Party Vendor Tool
- Combination of TOP or TOPAS and db2pd/PID cross-reference to Application Snapshot

IDUG 2009 Europe 19

NOTES: Just a short note on some techniques you can use to identify suboptimal SQL. After all, one of the hardest tasks sometimes is identifying SQL causing the problem. But there are several DB2 built-in tools and commands to help with this. Additionally, many third party vendor tools are also available.



## Suboptimal SQL (Date Function)

```
SELECT acc_num,    REAL_CHIPS,
       PROMO_CHIPS,    CASH_AMT
FROM DB2ADMIN.acc_balance
WHERE date(timestamp) = '2007-1-01' ;
```

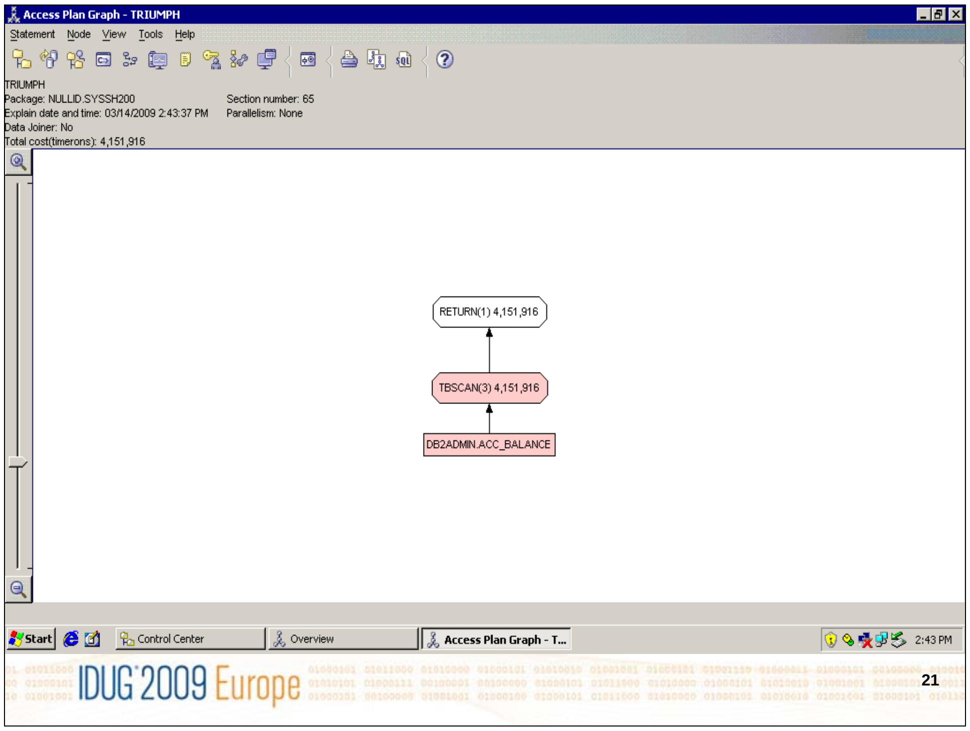
Timeron Cost:
4,151,916

No range-delimiting predicate

IDUG 2009 Europe

20

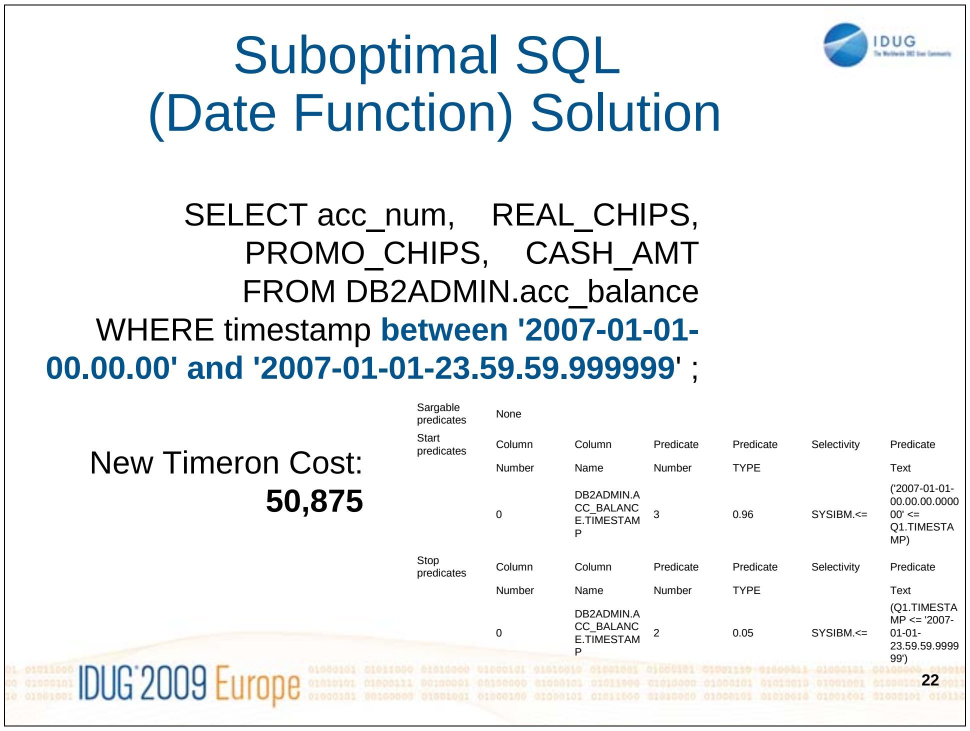
NOTES: This query is an example of a high cost query using a date function in the where clause. This query was caught using a GTS Top 10 SQL query. Note the timeron cost of over 4 million timerons.



TRIUMPH
Package: NULLID.SYSSH200
Explain date and time: 03/14/2009 2:43:37 PM
Data Joiner: No
Total cost(timerons): 4,151,916
Section number: 65
Parallelism: None

RETURN(1) 4,151,916
TBSCAN(3) 4,151,916
DB2ADMIN.ACC_BALANCE

NOTES: Using visual explain, you can see the high cost and the fact that a table scan is involved on the ACC_BALANCE table.



## Suboptimal SQL (Date Function) Solution

SELECT acc_num, REAL_CHIPS, PROMO_CHIPS, CASH_AMT FROM DB2ADMIN.acc_balance WHERE timestamp **between '2007-01-01-00.00.00' and '2007-01-01-23.59.59.999999'** ;

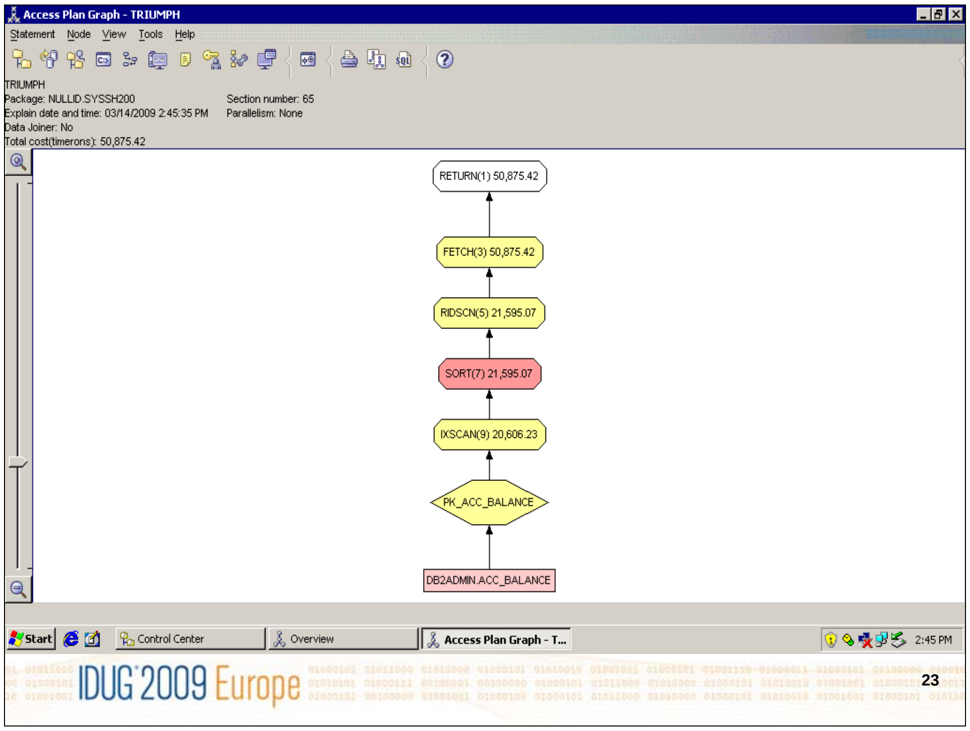
New Timeron Cost: **50,875**

| | | | | | | |
|---|---|---|---|---|---|---|
| Sargable predicates | None | | | | | |
| Start predicates | Column | Column | Predicate | Predicate | Selectivity | Predicate |
| | Number | Name | Number | TYPE | | Text |
| | 0 | DB2ADMIN.ACC_BALANCE.TIMESTAMP | 3 | 0.96 | SYSIBM.<= | ('2007-01-01-00.00.00.000000' <= Q1.TIMESTAMP) |
| Stop predicates | Column | Column | Predicate | Predicate | Selectivity | Predicate |
| | Number | Name | Number | TYPE | | Text |
| | 0 | DB2ADMIN.ACC_BALANCE.TIMESTAMP | 2 | 0.05 | SYSIBM.<= | (Q1.TIMESTAMP <= '2007-01-01-23.59.59.999999') |

IDUG 2009 Europe

22

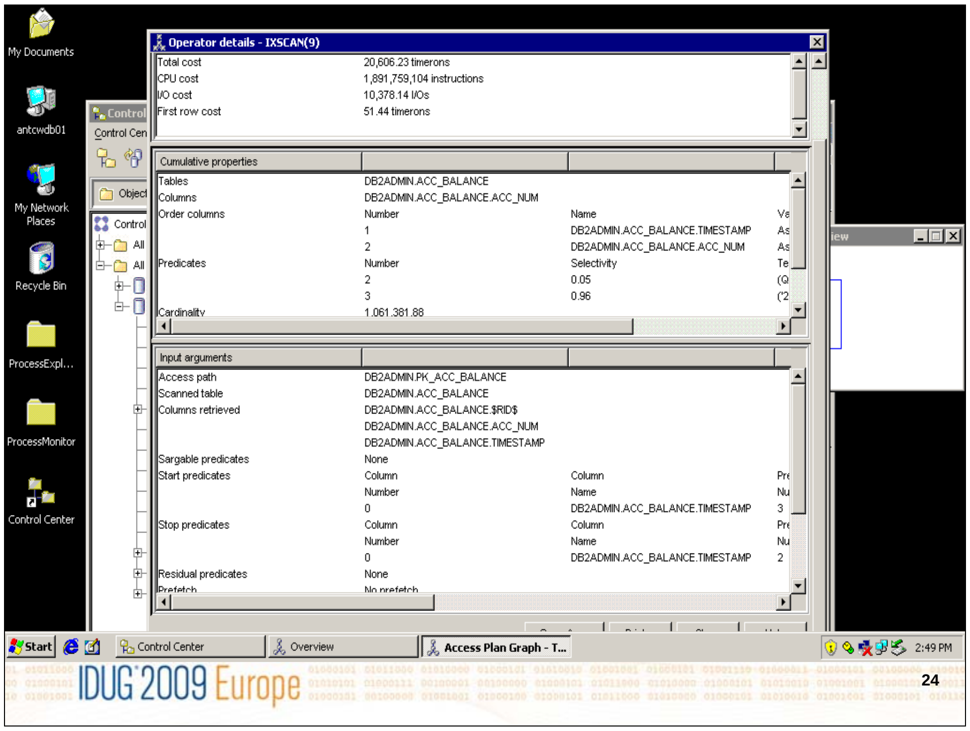
NOTES: By providing the specified date as a timestamp. The performance is enhanced because the values are used to start and stop range-delimiting predicates. Using a between predicate on the date range, the provided date range can be used as range-delimiting predicates and an existing index can be used. The result is improved response time and a huge improvement in cost.


Access Plan Graph - TRIUMPH

TRIUMPH
Package: NULLID.SYSSH200
Explain date and time: 03/14/2009 2:45:35 PM
Data Joiner: No
Total cost(timerons): 50,875.42

Section number: 65
Parallelism: None

NOTES: As you can observe, the new access plan is using the primary-key index to retrieve the columns specified. The table scan has been eliminated.



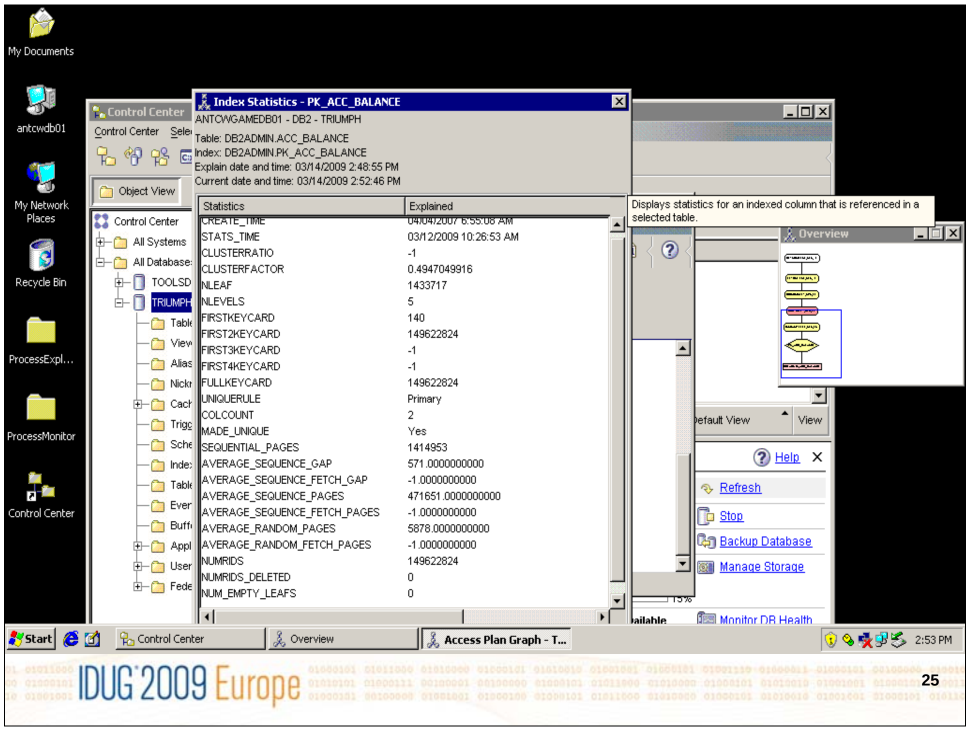
**Operator details - IXSCAN(9)**

| | |
|---|---|
| Total cost | 20,606.23 timerons |
| CPU cost | 1,891,759,104 instructions |
| I/O cost | 10,378.14 I/Os |
| First row cost | 51.44 timerons |

| Cumulative properties | | | |
|---|---|---|---|
| Tables | DB2ADMIN.ACC_BALANCE | | |
| Columns | DB2ADMIN.ACC_BALANCE.ACC_NUM | | |
| Order columns | Number | Name | Va |
| | 1 | DB2ADMIN.ACC_BALANCE.TIMESTAMP | As |
| | 2 | DB2ADMIN.ACC_BALANCE.ACC_NUM | As |
| Predicates | Number | Selectivity | Te |
| | 2 | 0.05 | (Q |
| | 3 | 0.96 | ('2 |
| Cardinality | 1,061,381.88 | | |

| Input arguments | | | |
|---|---|---|---|
| Access path | DB2ADMIN.PK_ACC_BALANCE | | |
| Scanned table | DB2ADMIN.ACC_BALANCE | | |
| Columns retrieved | DB2ADMIN.ACC_BALANCE.$RID$ | | |
| | DB2ADMIN.ACC_BALANCE.ACC_NUM | | |
| | DB2ADMIN.ACC_BALANCE.TIMESTAMP | | |
| Sargable predicates | None | | |
| Start predicates | Column | Column | Pre |
| | Number | Name | Nu |
| | 0 | DB2ADMIN.ACC_BALANCE.TIMESTAMP | 3 |
| Stop predicates | Column | Column | Pre |
| | Number | Name | Nu |
| | 0 | DB2ADMIN.ACC_BALANCE.TIMESTAMP | 2 |
| Residual predicates | None | | |
| Prefetch | No prefetch | | |

NOTES: In this slide, you can observe that range delimiting predicates are used via the visual explain drill down on the IXSCAN operator. You could also view similar information with db2exfmt.


Index Statistics - PK_ACC_BALANCE

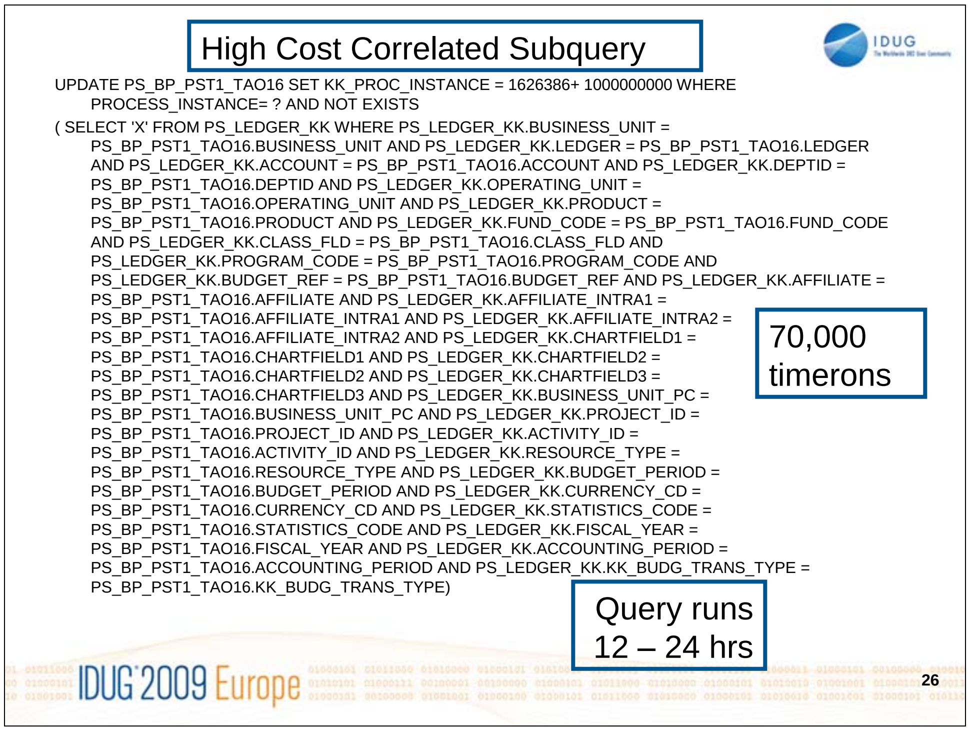
ANTCWGAMEDB01 - DB2 - TRIUMPH

Table: DB2ADMIN.ACC_BALANCE
Index: DB2ADMIN.PK_ACC_BALANCE
Explain date and time: 03/14/2009 2:48:55 PM
Current date and time: 03/14/2009 2:52:46 PM

| Statistics | Explained |
|---|---|
| CREATE_TIME | 04/04/2007 6:55:08 AM |
| STATS_TIME | 03/12/2009 10:26:53 AM |
| CLUSTERRATIO | -1 |
| CLUSTERFACTOR | 0.4947049916 |
| NLEAF | 1433717 |
| NLEVELS | 5 |
| FIRSTKEYCARD | 140 |
| FIRST2KEYCARD | 149622824 |
| FIRST3KEYCARD | -1 |
| FIRST4KEYCARD | -1 |
| FULLKEYCARD | 149622824 |
| UNIQUERULE | Primary |
| COLCOUNT | 2 |
| MADE_UNIQUE | Yes |
| SEQUENTIAL_PAGES | 1414953 |
| AVERAGE_SEQUENCE_GAP | 571.0000000000 |
| AVERAGE_SEQUENCE_FETCH_GAP | -1.0000000000 |
| AVERAGE_SEQUENCE_PAGES | 471651.0000000000 |
| AVERAGE_SEQUENCE_FETCH_PAGES | -1.0000000000 |
| AVERAGE_RANDOM_PAGES | 5878.0000000000 |
| AVERAGE_RANDOM_FETCH_PAGES | -1.0000000000 |
| NUMRIDS | 149622824 |
| NUMRIDS_DELETED | 0 |
| NUM_EMPTY_LEAFS | 0 |

Displays statistics for an indexed column that is referenced in a selected table.

NOTES: You can use visual explain to drill down further and review index or table statistics.



## High Cost Correlated Subquery

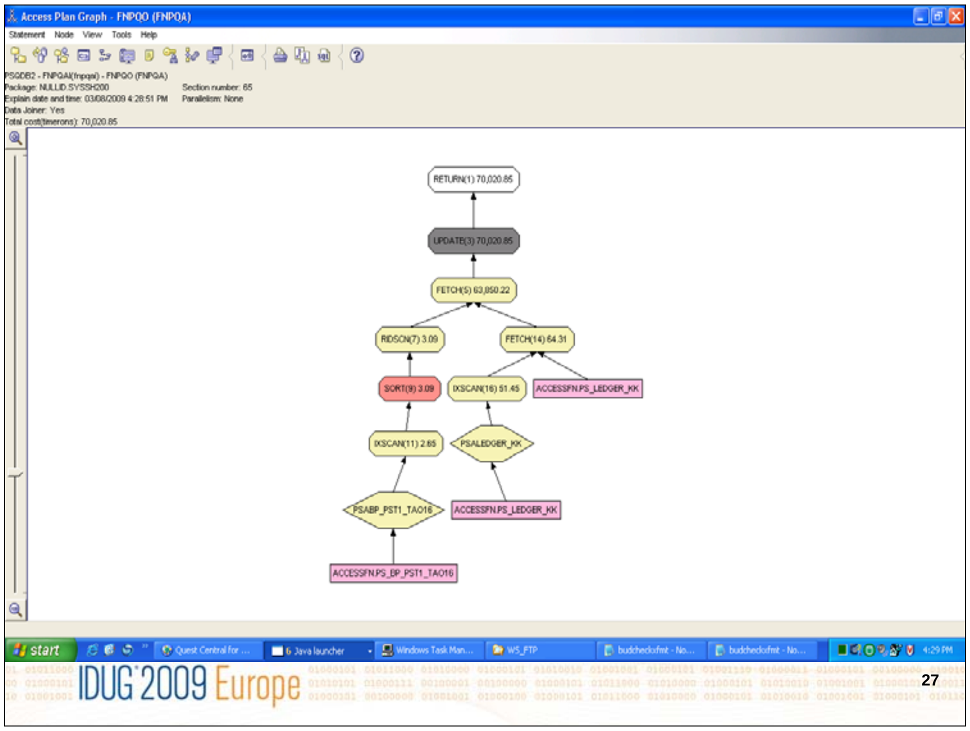
```
UPDATE PS_BP_PST1_TAO16 SET KK_PROC_INSTANCE = 1626386+ 1000000000 WHERE
      PROCESS_INSTANCE= ? AND NOT EXISTS
( SELECT 'X' FROM PS_LEDGER_KK WHERE PS_LEDGER_KK.BUSINESS_UNIT =
      PS_BP_PST1_TAO16.BUSINESS_UNIT AND PS_LEDGER_KK.LEDGER = PS_BP_PST1_TAO16.LEDGER
      AND PS_LEDGER_KK.ACCOUNT = PS_BP_PST1_TAO16.ACCOUNT AND PS_LEDGER_KK.DEPTID =
      PS_BP_PST1_TAO16.DEPTID AND PS_LEDGER_KK.OPERATING_UNIT =
      PS_BP_PST1_TAO16.OPERATING_UNIT AND PS_LEDGER_KK.PRODUCT =
      PS_BP_PST1_TAO16.PRODUCT AND PS_LEDGER_KK.FUND_CODE = PS_BP_PST1_TAO16.FUND_CODE
      AND PS_LEDGER_KK.CLASS_FLD = PS_BP_PST1_TAO16.CLASS_FLD AND
      PS_LEDGER_KK.PROGRAM_CODE = PS_BP_PST1_TAO16.PROGRAM_CODE AND
      PS_LEDGER_KK.BUDGET_REF = PS_BP_PST1_TAO16.BUDGET_REF AND PS_LEDGER_KK.AFFILIATE =
      PS_BP_PST1_TAO16.AFFILIATE AND PS_LEDGER_KK.AFFILIATE_INTRA1 =
      PS_BP_PST1_TAO16.AFFILIATE_INTRA1 AND PS_LEDGER_KK.AFFILIATE_INTRA2 =
      PS_BP_PST1_TAO16.AFFILIATE_INTRA2 AND PS_LEDGER_KK.CHARTFIELD1 =
      PS_BP_PST1_TAO16.CHARTFIELD1 AND PS_LEDGER_KK.CHARTFIELD2 =
      PS_BP_PST1_TAO16.CHARTFIELD2 AND PS_LEDGER_KK.CHARTFIELD3 =
      PS_BP_PST1_TAO16.CHARTFIELD3 AND PS_LEDGER_KK.BUSINESS_UNIT_PC =
      PS_BP_PST1_TAO16.BUSINESS_UNIT_PC AND PS_LEDGER_KK.PROJECT_ID =
      PS_BP_PST1_TAO16.PROJECT_ID AND PS_LEDGER_KK.ACTIVITY_ID =
      PS_BP_PST1_TAO16.ACTIVITY_ID AND PS_LEDGER_KK.RESOURCE_TYPE =
      PS_BP_PST1_TAO16.RESOURCE_TYPE AND PS_LEDGER_KK.BUDGET_PERIOD =
      PS_BP_PST1_TAO16.BUDGET_PERIOD AND PS_LEDGER_KK.CURRENCY_CD =
      PS_BP_PST1_TAO16.CURRENCY_CD AND PS_LEDGER_KK.STATISTICS_CODE =
      PS_BP_PST1_TAO16.STATISTICS_CODE AND PS_LEDGER_KK.FISCAL_YEAR =
      PS_BP_PST1_TAO16.FISCAL_YEAR AND PS_LEDGER_KK.ACCOUNTING_PERIOD =
      PS_BP_PST1_TAO16.ACCOUNTING_PERIOD AND PS_LEDGER_KK.KK_BUDG_TRANS_TYPE =
      PS_BP_PST1_TAO16.KK_BUDG_TRANS_TYPE)
```

70,000 timerons

Query runs 12 – 24 hrs

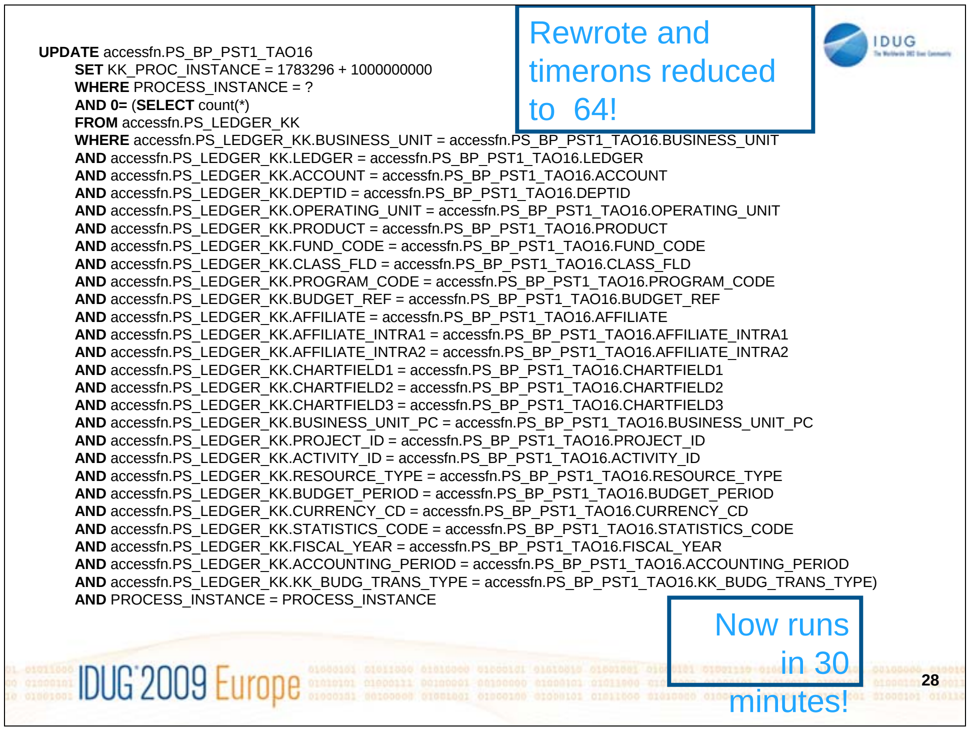
IDUG 2009 Europe 26

NOTES: This long running SQL was reported as "long-running" by the business end user when running a budget check PeopleSoft process. Analysis and discussion with business users revealed that budget check queries had historically been long-running and prone to lock timeouts. On a typical day, several of these could be running at the same time.



PSQDB2 - FNPQAI(fnpqai) - FNPQO (FNPQA)
Package: NULLID.SYSSH200 Section number: 65
Explain date and time: 03/08/2009 4:28:51 PM Parallelism: None
Data Joiner: Yes
Total cost(timerons): 70,020.85

NOTES: The SQL was captured via an application snapshot (and a third party vendor tool) and explained and analyzed. Indexes were being used on the PS_LEDGER_KK table and the TA016 (PeopleSoft real but temporary table) tables. The TA016 table could have anywhere between 0 and 60,000 rows depending on the budget check being run. The LEDGER table contains 10 million rows.



```
UPDATE accessfn.PS_BP_PST1_TAO16
       SET KK_PROC_INSTANCE = 1783296 + 1000000000
       WHERE PROCESS_INSTANCE = ?
       AND 0= (SELECT count(*)
       FROM accessfn.PS_LEDGER_KK
       WHERE accessfn.PS_LEDGER_KK.BUSINESS_UNIT = accessfn.PS_BP_PST1_TAO16.BUSINESS_UNIT
       AND accessfn.PS_LEDGER_KK.LEDGER = accessfn.PS_BP_PST1_TAO16.LEDGER
       AND accessfn.PS_LEDGER_KK.ACCOUNT = accessfn.PS_BP_PST1_TAO16.ACCOUNT
       AND accessfn.PS_LEDGER_KK.DEPTID = accessfn.PS_BP_PST1_TAO16.DEPTID
       AND accessfn.PS_LEDGER_KK.OPERATING_UNIT = accessfn.PS_BP_PST1_TAO16.OPERATING_UNIT
       AND accessfn.PS_LEDGER_KK.PRODUCT = accessfn.PS_BP_PST1_TAO16.PRODUCT
       AND accessfn.PS_LEDGER_KK.FUND_CODE = accessfn.PS_BP_PST1_TAO16.FUND_CODE
       AND accessfn.PS_LEDGER_KK.CLASS_FLD = accessfn.PS_BP_PST1_TAO16.CLASS_FLD
       AND accessfn.PS_LEDGER_KK.PROGRAM_CODE = accessfn.PS_BP_PST1_TAO16.PROGRAM_CODE
       AND accessfn.PS_LEDGER_KK.BUDGET_REF = accessfn.PS_BP_PST1_TAO16.BUDGET_REF
       AND accessfn.PS_LEDGER_KK.AFFILIATE = accessfn.PS_BP_PST1_TAO16.AFFILIATE
       AND accessfn.PS_LEDGER_KK.AFFILIATE_INTRA1 = accessfn.PS_BP_PST1_TAO16.AFFILIATE_INTRA1
       AND accessfn.PS_LEDGER_KK.AFFILIATE_INTRA2 = accessfn.PS_BP_PST1_TAO16.AFFILIATE_INTRA2
       AND accessfn.PS_LEDGER_KK.CHARTFIELD1 = accessfn.PS_BP_PST1_TAO16.CHARTFIELD1
       AND accessfn.PS_LEDGER_KK.CHARTFIELD2 = accessfn.PS_BP_PST1_TAO16.CHARTFIELD2
       AND accessfn.PS_LEDGER_KK.CHARTFIELD3 = accessfn.PS_BP_PST1_TAO16.CHARTFIELD3
       AND accessfn.PS_LEDGER_KK.BUSINESS_UNIT_PC = accessfn.PS_BP_PST1_TAO16.BUSINESS_UNIT_PC
       AND accessfn.PS_LEDGER_KK.PROJECT_ID = accessfn.PS_BP_PST1_TAO16.PROJECT_ID
       AND accessfn.PS_LEDGER_KK.ACTIVITY_ID = accessfn.PS_BP_PST1_TAO16.ACTIVITY_ID
       AND accessfn.PS_LEDGER_KK.RESOURCE_TYPE = accessfn.PS_BP_PST1_TAO16.RESOURCE_TYPE
       AND accessfn.PS_LEDGER_KK.BUDGET_PERIOD = accessfn.PS_BP_PST1_TAO16.BUDGET_PERIOD
       AND accessfn.PS_LEDGER_KK.CURRENCY_CD = accessfn.PS_BP_PST1_TAO16.CURRENCY_CD
       AND accessfn.PS_LEDGER_KK.STATISTICS_CODE = accessfn.PS_BP_PST1_TAO16.STATISTICS_CODE
       AND accessfn.PS_LEDGER_KK.FISCAL_YEAR = accessfn.PS_BP_PST1_TAO16.FISCAL_YEAR
       AND accessfn.PS_LEDGER_KK.ACCOUNTING_PERIOD = accessfn.PS_BP_PST1_TAO16.ACCOUNTING_PERIOD
       AND accessfn.PS_LEDGER_KK.KK_BUDG_TRANS_TYPE = accessfn.PS_BP_PST1_TAO16.KK_BUDG_TRANS_TYPE)
       AND PROCESS_INSTANCE = PROCESS_INSTANCE
```

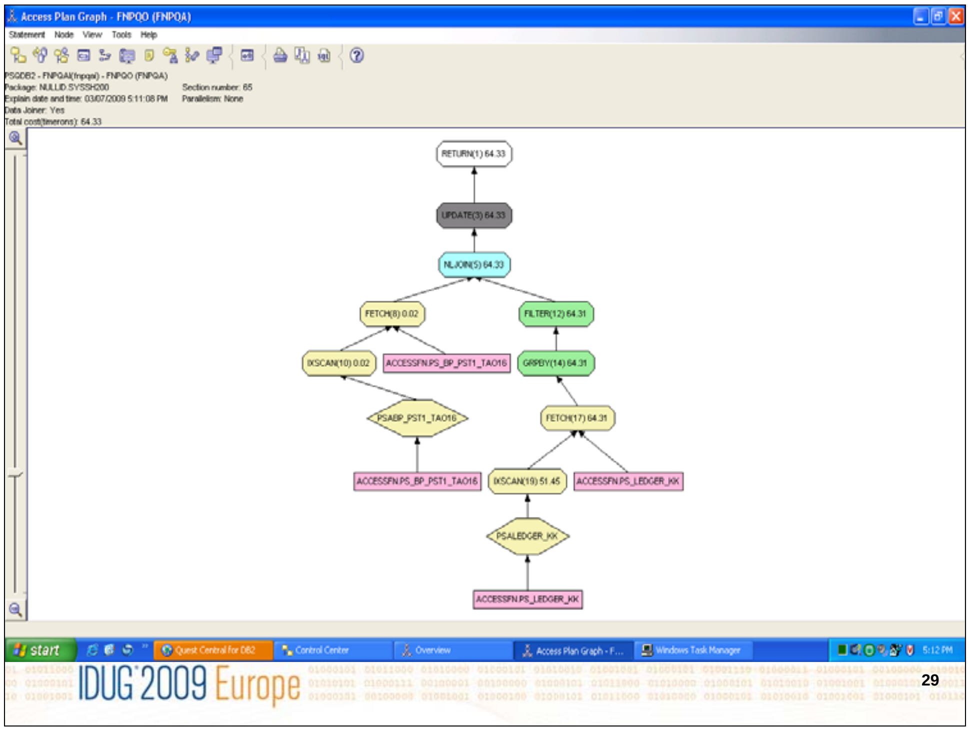
Rewrote and timerons reduced to 64!

Now runs in 30 minutes!

IDUG 2009 Europe

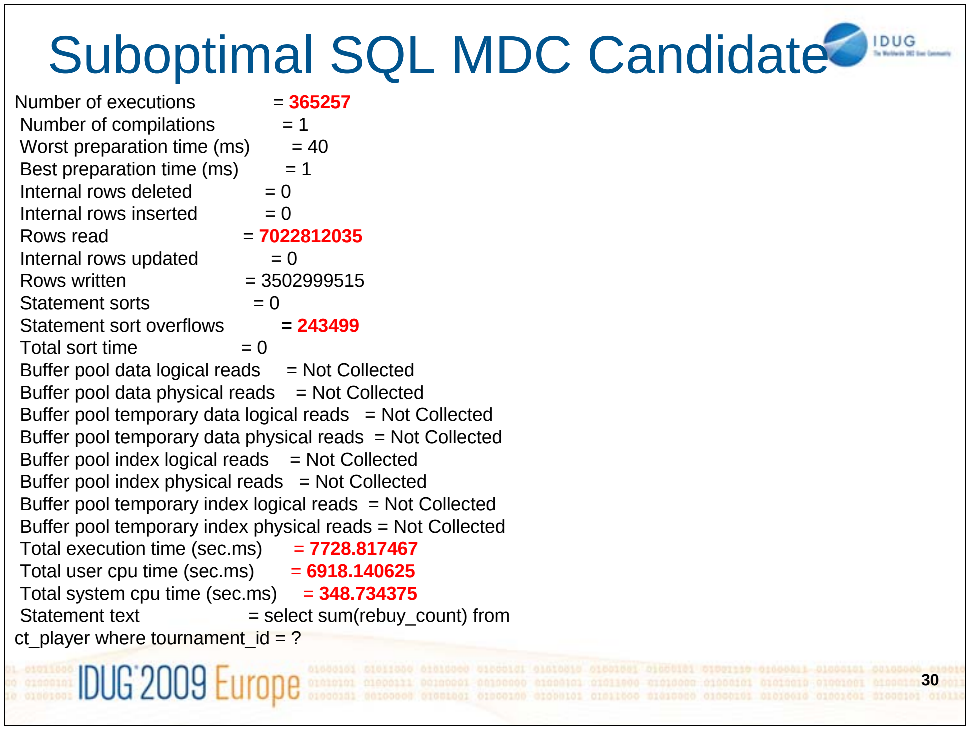
28

NOTES: This is the rewritten query from previous slide. This rewrite reduced elapsed time from 12 – 24 hrs to 30-40 minutes, enabling the company to run these much more often and in accordance with their business needs. There were no more reports it being long running or anymore lock timeouts. This solution has been implemented and running for over a year and provides consistent performance.



```
PSGDB2 - FNPQA(fnpqa) - FNPQO (FNPQA)
Package: NULLID.SYSSH200                        Section number: 65
Explain date and time: 03/07/2009 5:11:08 PM    Parallelism: None
Data Joiner: Yes
Total cost(timerons): 64.33
```

NOTES: After the SQL was rewritten, cost is only 64 timerons. Another issue complicating this query is that the temp table sometimes has statistics on it and sometimes it doesn’t, thence this SQL was also subject to changing access paths. The table was therefore marked as “volatile” and after trial and error, it was found that it was best to not run runstats on the TA016 table.

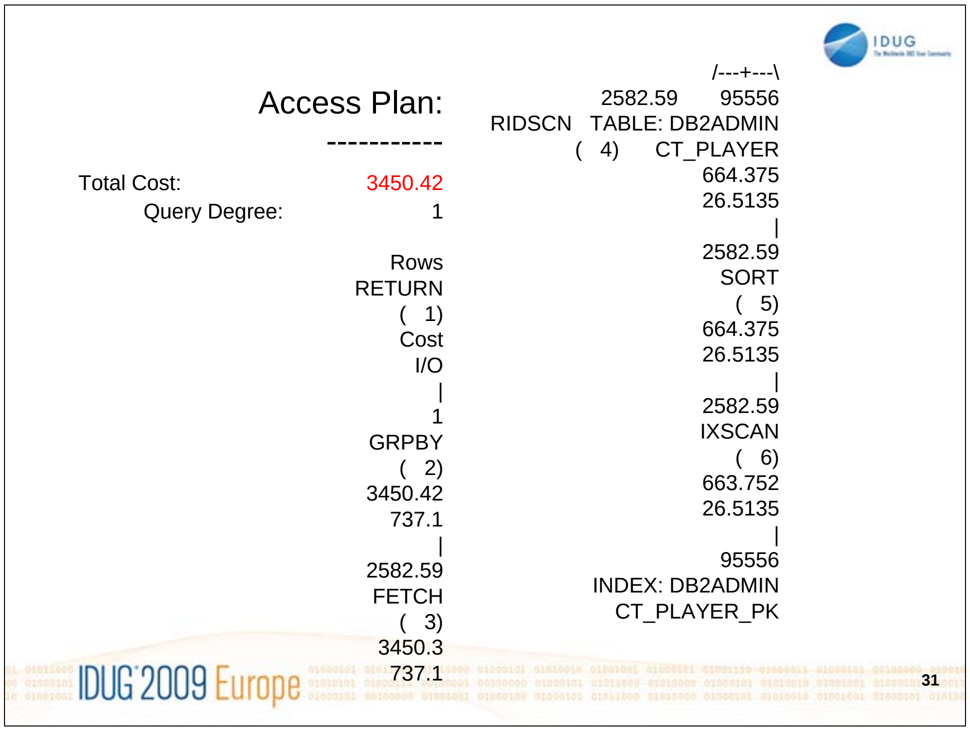


## Suboptimal SQL MDC Candidate

```
Number of executions              = 365257
 Number of compilations             = 1
 Worst preparation time (ms)        = 40
 Best preparation time (ms)         = 1
 Internal rows deleted            = 0
 Internal rows inserted           = 0
 Rows read                   = 7022812035
 Internal rows updated              = 0
 Rows written                  = 3502999515
 Statement sorts                  = 0
 Statement sort overflows           = 243499
 Total sort time                 = 0
 Buffer pool data logical reads     = Not Collected
 Buffer pool data physical reads    = Not Collected
 Buffer pool temporary data logical reads   = Not Collected
 Buffer pool temporary data physical reads  = Not Collected
 Buffer pool index logical reads    = Not Collected
 Buffer pool index physical reads   = Not Collected
 Buffer pool temporary index logical reads  = Not Collected
 Buffer pool temporary index physical reads = Not Collected
 Total execution time (sec.ms)     = 7728.817467
 Total user cpu time (sec.ms)      = 6918.140625
 Total system cpu time (sec.ms)     = 348.734375
 Statement text                  = select sum(rebuy_count) from
ct_player where tournament_id = ?
```

IDUG 2009 Europe 30

NOTES: This SQL became a candidate for an MDC table because of its very high frequency and use of list prefetch. This SQL was reviewed before the Design Advisor had been enhanced to support recommendations for MDC tables. MDC tables are tables that can be clustered on multiple dimensions. MDC tables use Block indexes which point to blocks. This is unlike RID indexes which point to rows. Hence Block Indexes are smaller and are very good at speeding up list prefetch and range scans, as the Block Index can be scanned.



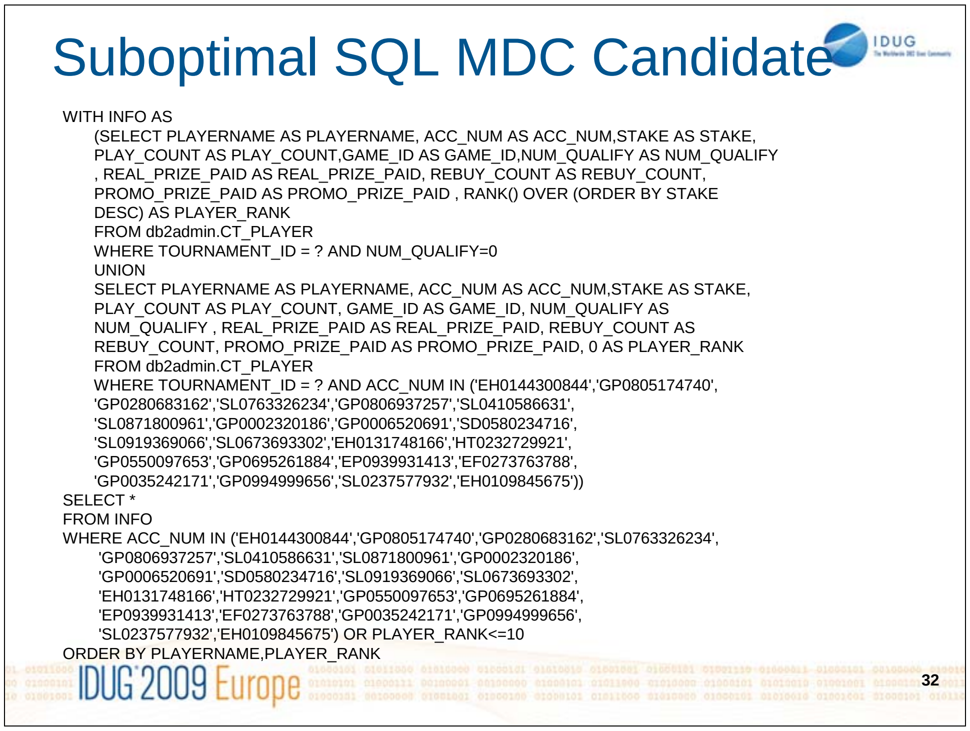
```
Access Plan:
-----------
Total Cost:         3450.42
Query Degree:             1

                       Rows
                     RETURN
                     (   1)
                       Cost
                        I/O
                          |
                          1
                      GRPBY
                     (   2)
                    3450.42
                      737.1
                          |
                    2582.59
                      FETCH
                     (   3)
                     3450.3
                      737.1
```

```
             /---+---\
     2582.59       95556
RIDSCN   TABLE: DB2ADMIN
(   4)         CT_PLAYER
 664.375
 26.5135
    |
 2582.59
  SORT
 (   5)
 664.375
 26.5135
    |
 2582.59
 IXSCAN
 (   6)
 663.752
 26.5135
    |
  95556
INDEX: DB2ADMIN
CT_PLAYER_PK
```

IDUG 2009 Europe

31

NOTES: This is the access plan before the MDC table was created.

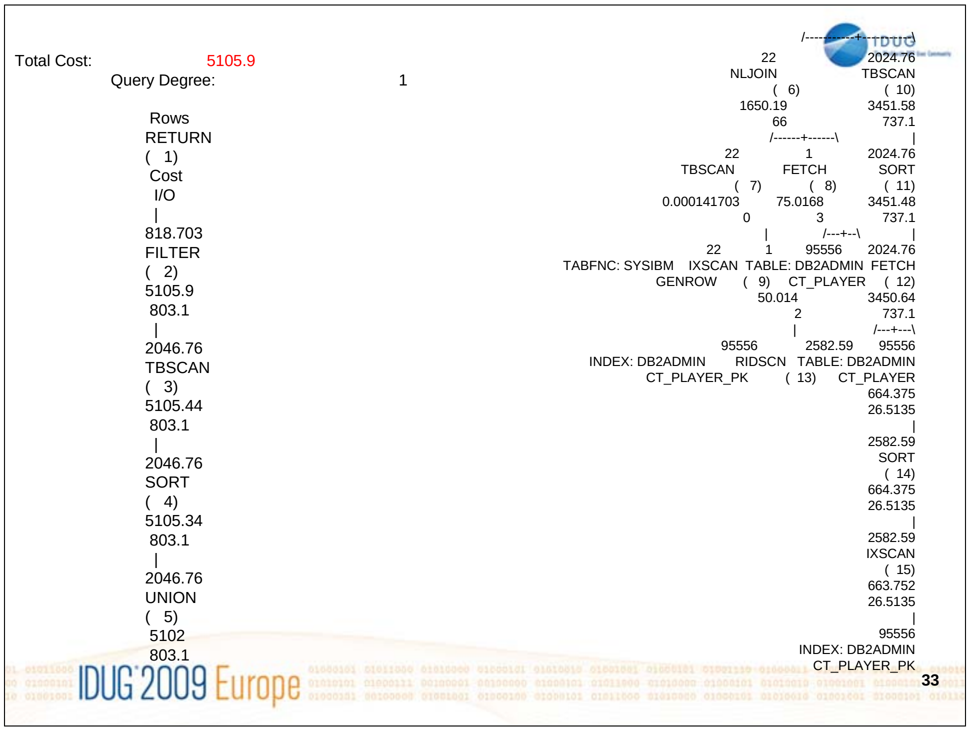


## Suboptimal SQL MDC Candidate

```
WITH INFO AS
      (SELECT PLAYERNAME AS PLAYERNAME, ACC_NUM AS ACC_NUM,STAKE AS STAKE,
      PLAY_COUNT AS PLAY_COUNT,GAME_ID AS GAME_ID,NUM_QUALIFY AS NUM_QUALIFY
      , REAL_PRIZE_PAID AS REAL_PRIZE_PAID, REBUY_COUNT AS REBUY_COUNT,
      PROMO_PRIZE_PAID AS PROMO_PRIZE_PAID , RANK() OVER (ORDER BY STAKE
      DESC) AS PLAYER_RANK
      FROM db2admin.CT_PLAYER
      WHERE TOURNAMENT_ID = ? AND NUM_QUALIFY=0
      UNION
      SELECT PLAYERNAME AS PLAYERNAME, ACC_NUM AS ACC_NUM,STAKE AS STAKE,
      PLAY_COUNT AS PLAY_COUNT, GAME_ID AS GAME_ID, NUM_QUALIFY AS
      NUM_QUALIFY , REAL_PRIZE_PAID AS REAL_PRIZE_PAID, REBUY_COUNT AS
      REBUY_COUNT, PROMO_PRIZE_PAID AS PROMO_PRIZE_PAID, 0 AS PLAYER_RANK
      FROM db2admin.CT_PLAYER
      WHERE TOURNAMENT_ID = ? AND ACC_NUM IN ('EH0144300844','GP0805174740',
      'GP0280683162','SL0763326234','GP0806937257','SL0410586631',
      'SL0871800961','GP0002320186','GP0006520691','SD0580234716',
      'SL0919369066','SL0673693302','EH0131748166','HT0232729921',
      'GP0550097653','GP0695261884','EP0939931413','EF0273763788',
      'GP0035242171','GP0994999656','SL0237577932','EH0109845675'))
SELECT *
FROM INFO
WHERE ACC_NUM IN ('EH0144300844','GP0805174740','GP0280683162','SL0763326234',
       'GP0806937257','SL0410586631','SL0871800961','GP0002320186',
       'GP0006520691','SD0580234716','SL0919369066','SL0673693302',
       'EH0131748166','HT0232729921','GP0550097653','GP0695261884',
       'EP0939931413','EF0273763788','GP0035242171','GP0994999656',
       'SL0237577932','EH0109845675') OR PLAYER_RANK<=10
ORDER BY PLAYERNAME,PLAYER_RANK
```

32

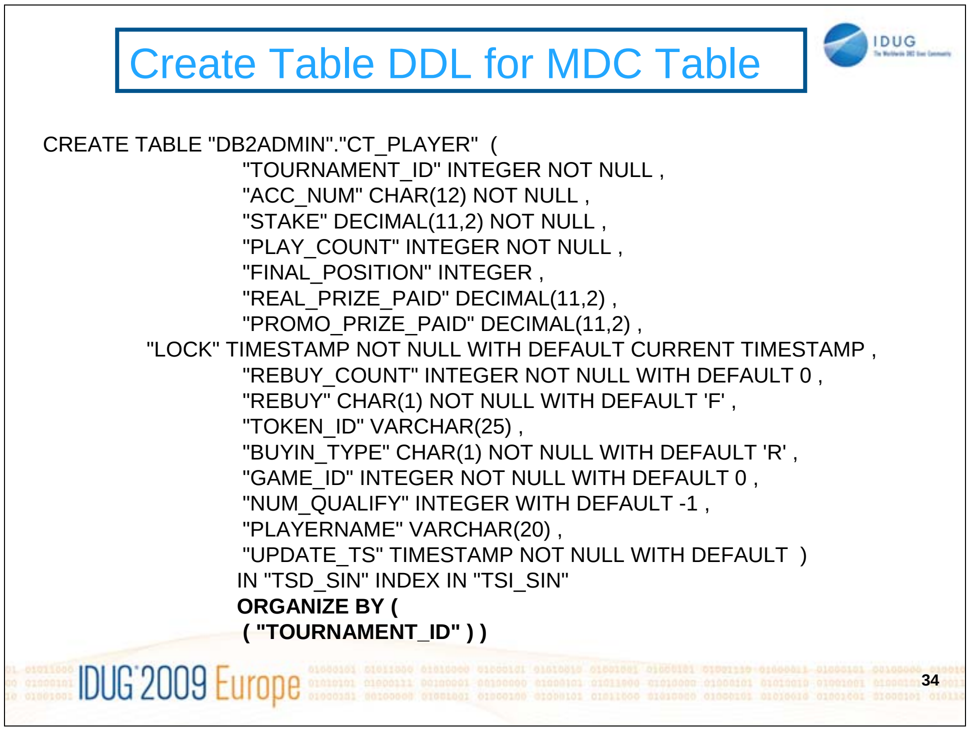
NOTES: This SQL was captured using a Top 10 SQL snapshot function. This Common Table Expression was also executed with high frequency.



```
Total Cost:             5105.9
        Query Degree:                 1

              Rows
             RETURN
             (   1)
              Cost
               I/O
               |
             818.703
             FILTER
             (   2)
             5105.9
              803.1
               |
             2046.76
             TBSCAN
             (   3)
             5105.44
              803.1
               |
             2046.76
             SORT
             (   4)
             5105.34
              803.1
               |
             2046.76
             UNION
             (   5)
              5102
              803.1
```

```
                                   /----------+----------\
                              22                      2024.76
                            NLJOIN                     TBSCAN
                            (   6)                     (  10)
                           1650.19                    3451.58
                              66                        737.1
                        /------+------\                     |
                      22              1               2024.76
                    TBSCAN          FETCH                SORT
                    (   7)          (   8)             (  11)
              0.000141703        75.0168              3451.48
                        0              3                737.1
                        |           /---+--\                |
                   22          1     95556            2024.76
TABFNC: SYSIBM    IXSCAN  TABLE: DB2ADMIN               FETCH
          GENROW      (   9)    CT_PLAYER              (  12)
                      50.014                          3450.64
                           2                            737.1
                           |                         /---+---\
             95556          2582.59                     95556
INDEX: DB2ADMIN       RIDSCN   TABLE: DB2ADMIN
   CT_PLAYER_PK       (  13)     CT_PLAYER
                                                      664.375
                                                      26.5135
                                                            |
                                                      2582.59
                                                         SORT
                                                       (  14)
                                                      664.375
                                                      26.5135
                                                            |
                                                      2582.59
                                                       IXSCAN
                                                       (  15)
                                                      663.752
                                                      26.5135
                                                            |
                                                        95556
                                              INDEX: DB2ADMIN
                                                 CT_PLAYER_PK
```

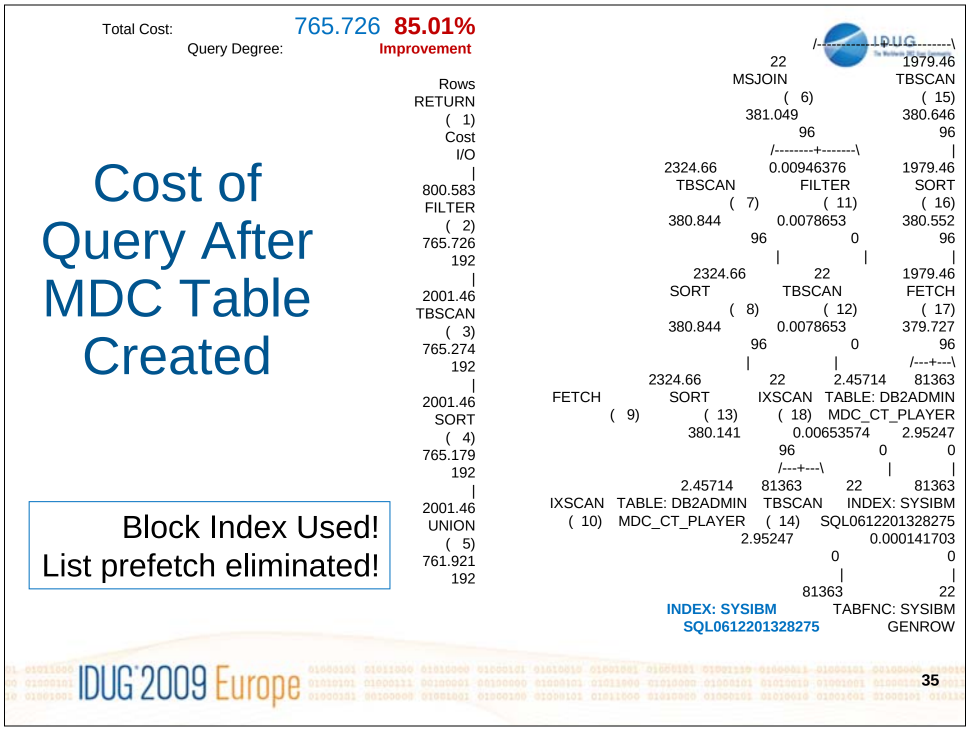
NOTES: Both MDC candidate queries were captured via dynamic sql snapshots and the Top 10 query, and both were going against the same table. These queries are used to rank online gamers in some of the popular sporting online games.


## Create Table DDL for MDC Table

```
CREATE TABLE "DB2ADMIN"."CT_PLAYER"  (
                "TOURNAMENT_ID" INTEGER NOT NULL ,
                "ACC_NUM" CHAR(12) NOT NULL ,
                "STAKE" DECIMAL(11,2) NOT NULL ,
                "PLAY_COUNT" INTEGER NOT NULL ,
                "FINAL_POSITION" INTEGER ,
                "REAL_PRIZE_PAID" DECIMAL(11,2) ,
                "PROMO_PRIZE_PAID" DECIMAL(11,2) ,
        "LOCK" TIMESTAMP NOT NULL WITH DEFAULT CURRENT TIMESTAMP ,
                "REBUY_COUNT" INTEGER NOT NULL WITH DEFAULT 0 ,
                "REBUY" CHAR(1) NOT NULL WITH DEFAULT 'F' ,
                "TOKEN_ID" VARCHAR(25) ,
                "BUYIN_TYPE" CHAR(1) NOT NULL WITH DEFAULT 'R' ,
                "GAME_ID" INTEGER NOT NULL WITH DEFAULT 0 ,
                "NUM_QUALIFY" INTEGER WITH DEFAULT -1 ,
                "PLAYERNAME" VARCHAR(20) ,
                "UPDATE_TS" TIMESTAMP NOT NULL WITH DEFAULT  )
               IN "TSD_SIN" INDEX IN "TSI_SIN"
               ORGANIZE BY (
                ( "TOURNAMENT_ID" ) )
```

IDUG 2009 Europe 34

NOTES: This slide contains DDL for creating the MDC table that was used to improve the SQL. Note that this MDC table was used in an high volume OLTP database. MDC can provide huge benefits in OLTP environments in addition to DW/BI environments. This is because most databases are not pure OLTP type databases but are a mixture of OLTP and Hybrid queries. Also, as of DB2 V8.2, Design Advisor will recommend MDC tables if requested.



Total Cost: 765.726 **85.01%**
Query Degree: **Improvement**

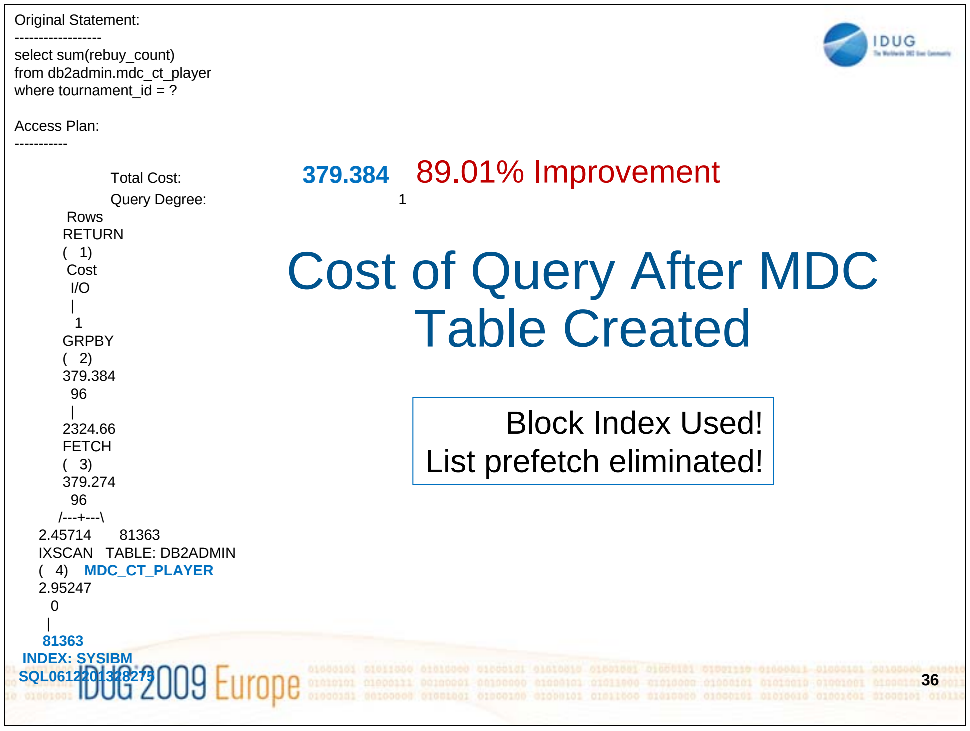
## Cost of Query After MDC Table Created

Block Index Used!
List prefetch eliminated!

```
      Rows
    RETURN
    (   1)
      Cost
       I/O
         |
   800.583
    FILTER
    (   2)
   765.726
       192
         |
   2001.46
    TBSCAN
    (   3)
   765.274
       192
         |
   2001.46
      SORT
    (   4)
   765.179
       192
         |
   2001.46
     UNION
    (   5)
   761.921
       192
```

```
                                      /-------------+-------------\
                                    22                          1979.46
                                 MSJOIN                          TBSCAN
                                 (   6)                          (  15)
                                381.049                          380.646
                                     96                               96
                              /--------+-------\                      |
          2324.66          0.00946376                            1979.46
           TBSCAN             FILTER                                SORT
          (   7)             (  11)                              (  16)
          380.844          0.0078653                             380.552
               96                  0                                  96
                |                  |                                   |
          2324.66                 22                             1979.46
             SORT             TBSCAN                               FETCH
          (   8)             (  12)                              (  17)
          380.844          0.0078653                             379.727
               96                  0                                  96
                |                  |                           /---+---\
          2324.66                 22         2.45714              81363
FETCH        SORT             IXSCAN   TABLE: DB2ADMIN
(   9)     (  13)             (  18)     MDC_CT_PLAYER
           380.141         0.00653574        2.95247
                                  96              0                    0
                             /---+---\            |                    |
   2.45714          81363         22                               81363
IXSCAN   TABLE: DB2ADMIN    TBSCAN                        INDEX: SYSIBM
(  10)     MDC_CT_PLAYER    (  14)                    SQL0612201328275
                             2.95247                          0.000141703
                                   0                                   0
                                   |                                   |
                               81363                                  22
        INDEX: SYSIBM                                    TABFNC: SYSIBM
     SQL0612201328275                                            GENROW
```

NOTES: Note that the block index is used and list prefetch eliminated. Additionally, block indexes are smaller and may be more resident in the bufferpool.



## Cost of Query After MDC Table Created

```
Original Statement:
------------------
select sum(rebuy_count)
from db2admin.mdc_ct_player
where tournament_id = ?

Access Plan:
-----------

             Total Cost:              379.384
             Query Degree:                  1
         Rows
        RETURN
        (   1)
         Cost
          I/O
           |
            1
        GRPBY
        (   2)
        379.384
          96
           |
        2324.66
        FETCH
        (   3)
        379.274
          96
       /---+---\
  2.45714      81363
  IXSCAN   TABLE: DB2ADMIN
  (   4)   MDC_CT_PLAYER
  2.95247
     0
    |
   81363
 INDEX: SYSIBM
SQL061220132827S
```

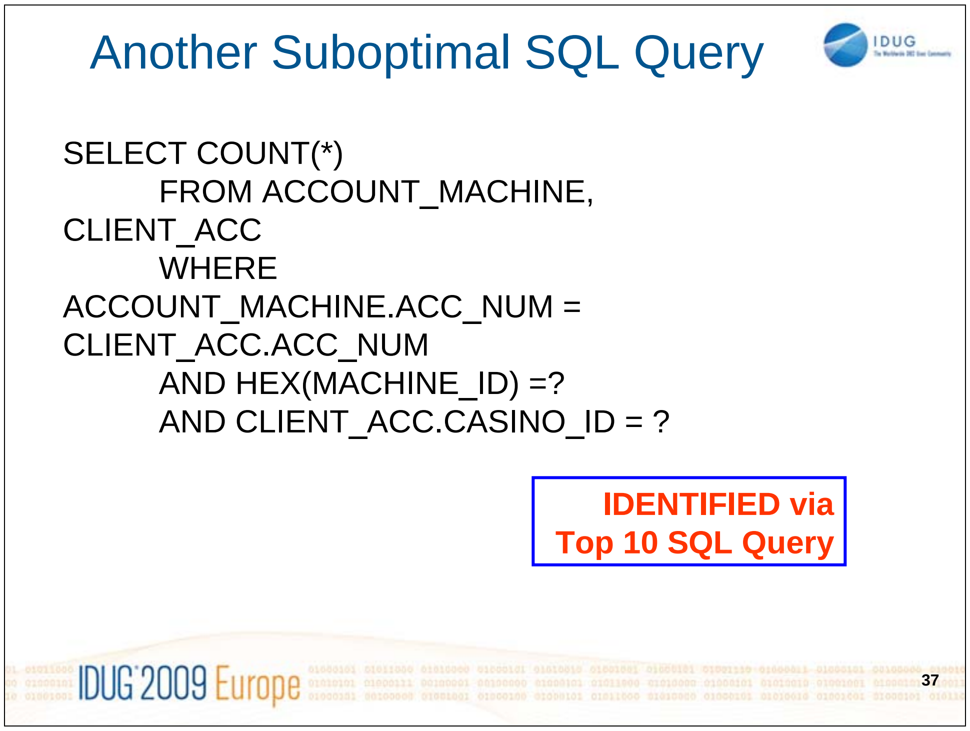
89.01% Improvement

Block Index Used!
List prefetch eliminated!

IDUG 2009 Europe

36

NOTES: This was a high priority query which is executed many times a second. The reduction in timerons and response time resulted in drastic improvement. As in the previous case the block index is used.



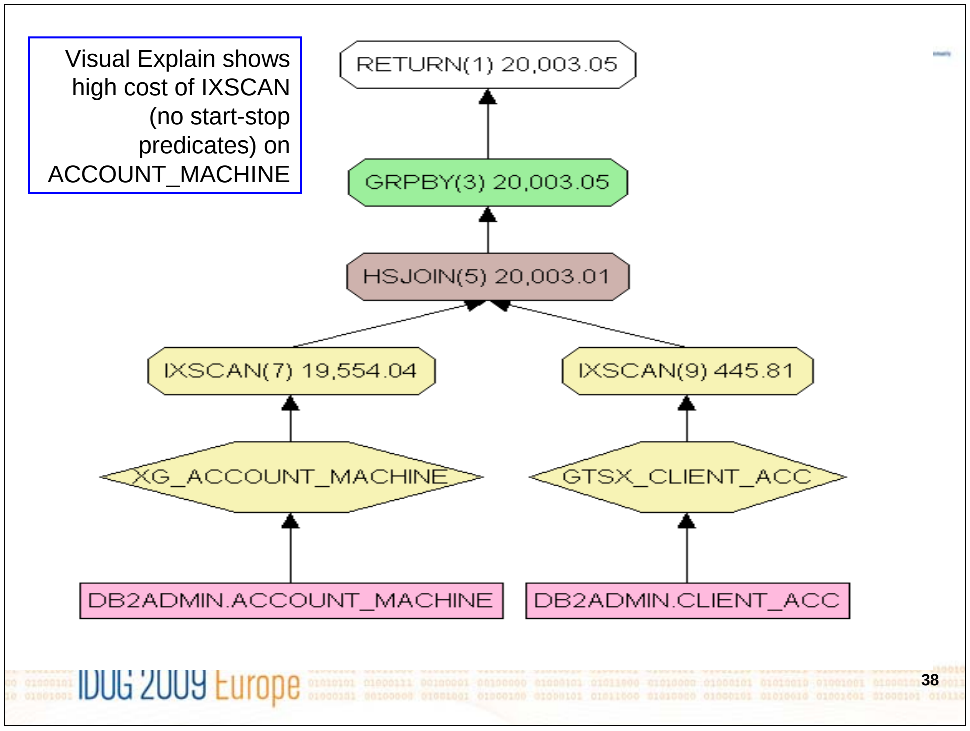
## Another Suboptimal SQL Query

```
SELECT COUNT(*)
        FROM ACCOUNT_MACHINE,
CLIENT_ACC
        WHERE
ACCOUNT_MACHINE.ACC_NUM =
CLIENT_ACC.ACC_NUM
        AND HEX(MACHINE_ID) =?
        AND CLIENT_ACC.CASINO_ID = ?
```

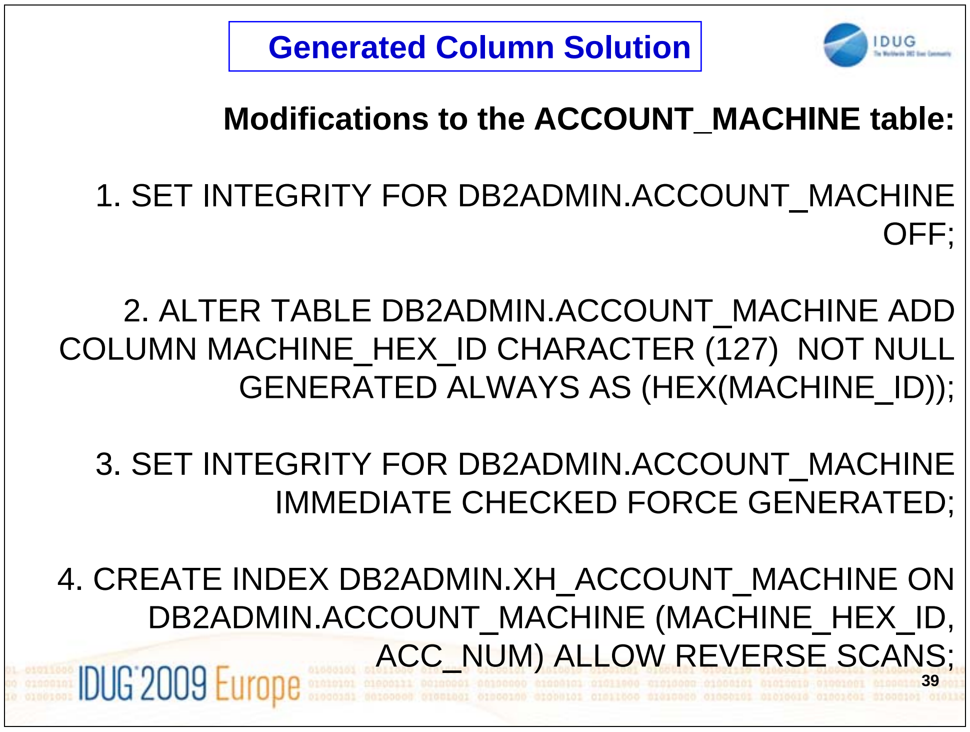
**IDENTIFIED via Top 10 SQL Query**

IDUG 2009 Europe 37

NOTES: This suboptimal SQL was captured with a Top 10 snapshot SQL function that ranks dynamic SQL in terms of SYSTEM CPU, USER CPU, ROWS READ, etc.  The Hex function precluded range delimiting predicate from being used.


Visual Explain shows high cost of IXSCAN (no start-stop predicates) on ACCOUNT_MACHINE

NOTES: Visual explain of high cost SQL HEX function.

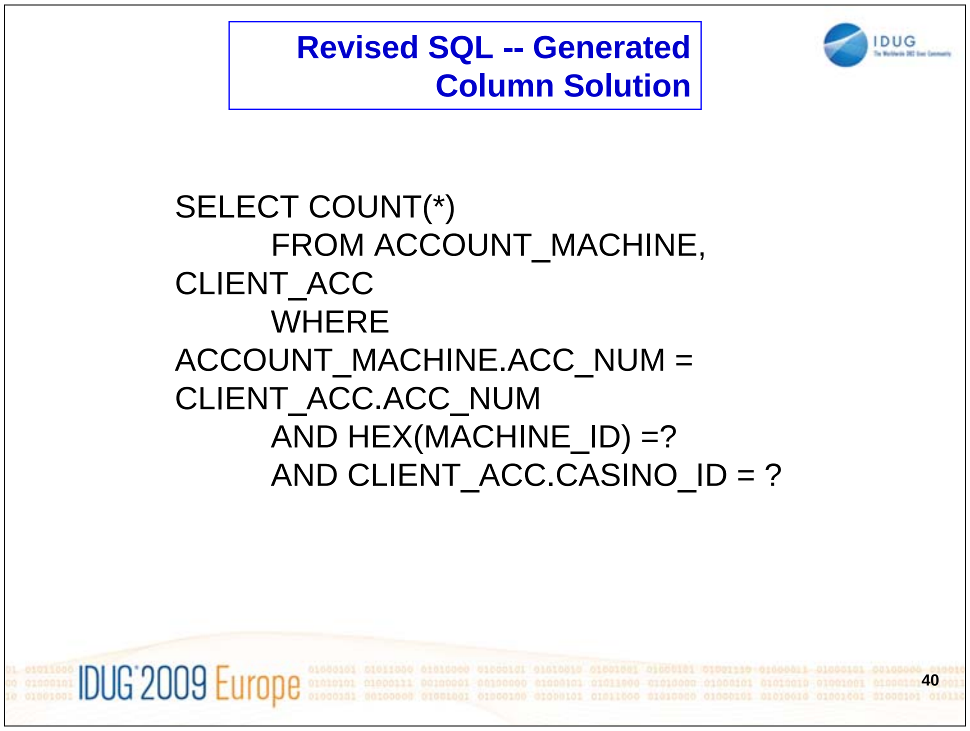


## Generated Column Solution

**Modifications to the ACCOUNT_MACHINE table:**

```
1. SET INTEGRITY FOR DB2ADMIN.ACCOUNT_MACHINE
OFF;

2. ALTER TABLE DB2ADMIN.ACCOUNT_MACHINE ADD
COLUMN MACHINE_HEX_ID CHARACTER (127)  NOT NULL
GENERATED ALWAYS AS (HEX(MACHINE_ID));

3. SET INTEGRITY FOR DB2ADMIN.ACCOUNT_MACHINE
IMMEDIATE CHECKED FORCE GENERATED;

4. CREATE INDEX DB2ADMIN.XH_ACCOUNT_MACHINE ON
DB2ADMIN.ACCOUNT_MACHINE (MACHINE_HEX_ID,
ACC_NUM) ALLOW REVERSE SCANS;
```

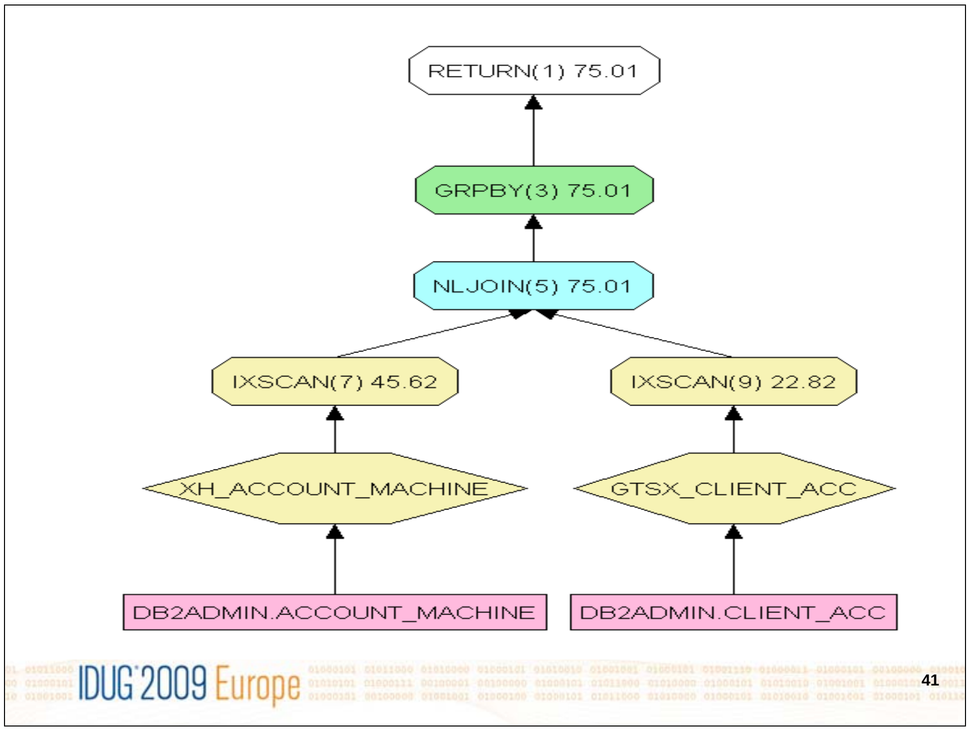
NOTES: A good solution to this problem is to use a generated column. In other words, use the DB2 engine to generate the column as HEX upon insertion and create an index on the generated column. This was done as shown on the slide.



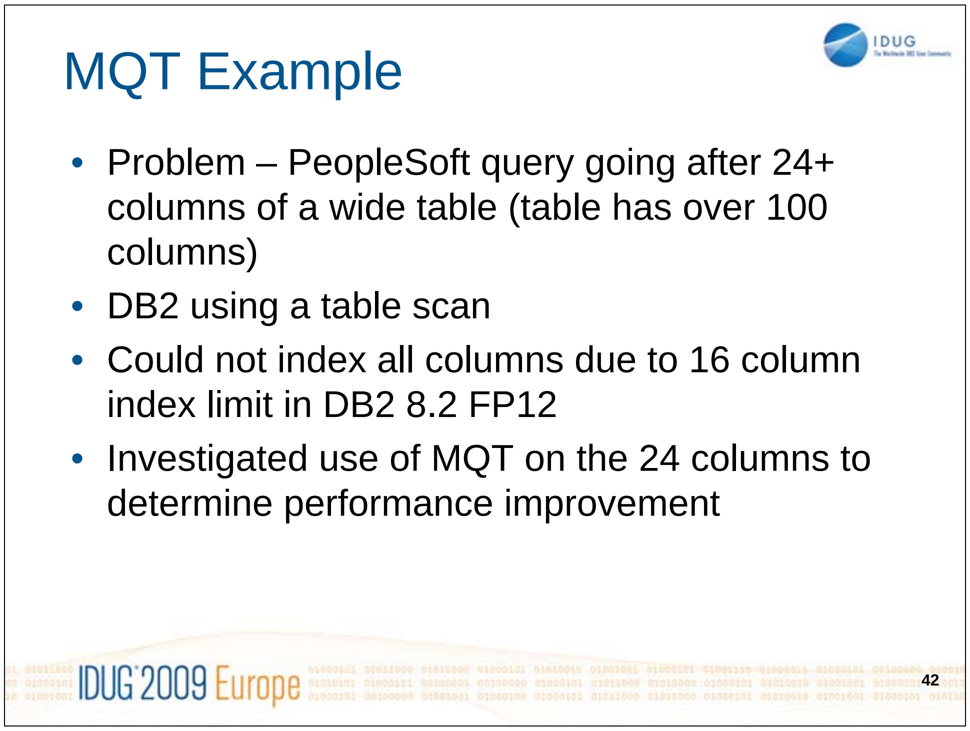
## Revised SQL -- Generated Column Solution

```
SELECT COUNT(*)
        FROM ACCOUNT_MACHINE,
CLIENT_ACC
        WHERE
ACCOUNT_MACHINE.ACC_NUM =
CLIENT_ACC.ACC_NUM
        AND HEX(MACHINE_ID) =?
        AND CLIENT_ACC.CASINO_ID = ?
```

NOTES: The SQL was rewritten to remove the HEX function and make use of the generated column.



NOTES: Visual explain after SQL rewritten, generated column created, and new index created on generated column.  Huge performance improvement realized and query no longer in the top 10 SQL report.



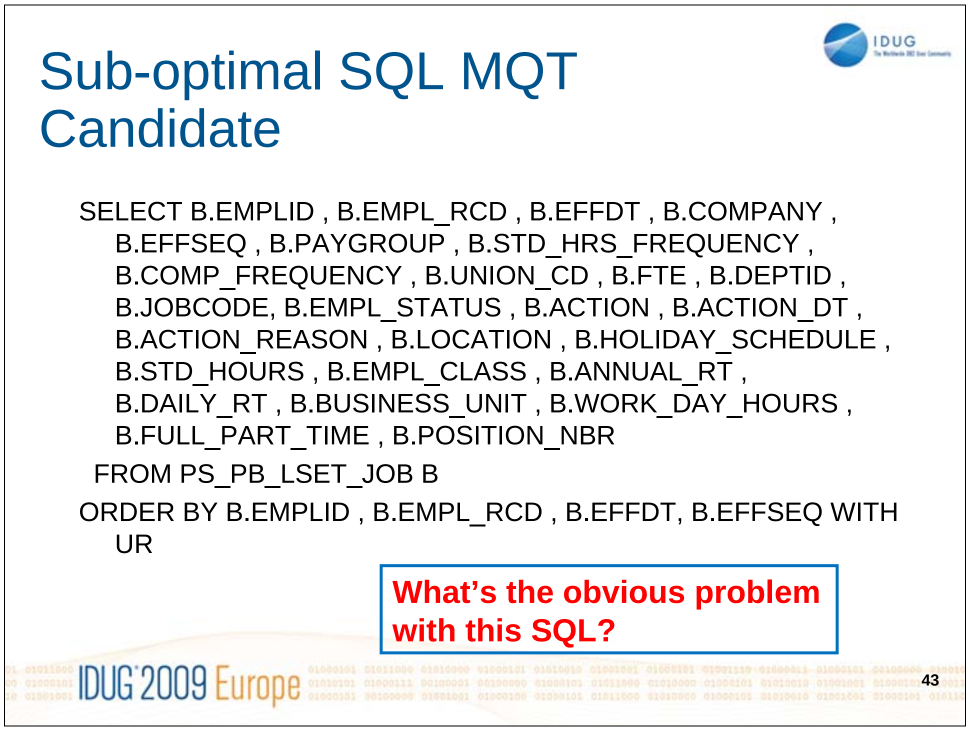
## MQT Example

- Problem – PeopleSoft query going after 24+ columns of a wide table (table has over 100 columns)
- DB2 using a table scan
- Could not index all columns due to 16 column index limit in DB2 8.2 FP12
- Investigated use of MQT on the 24 columns to determine performance improvement

IDUG 2009 Europe

42

NOTES:

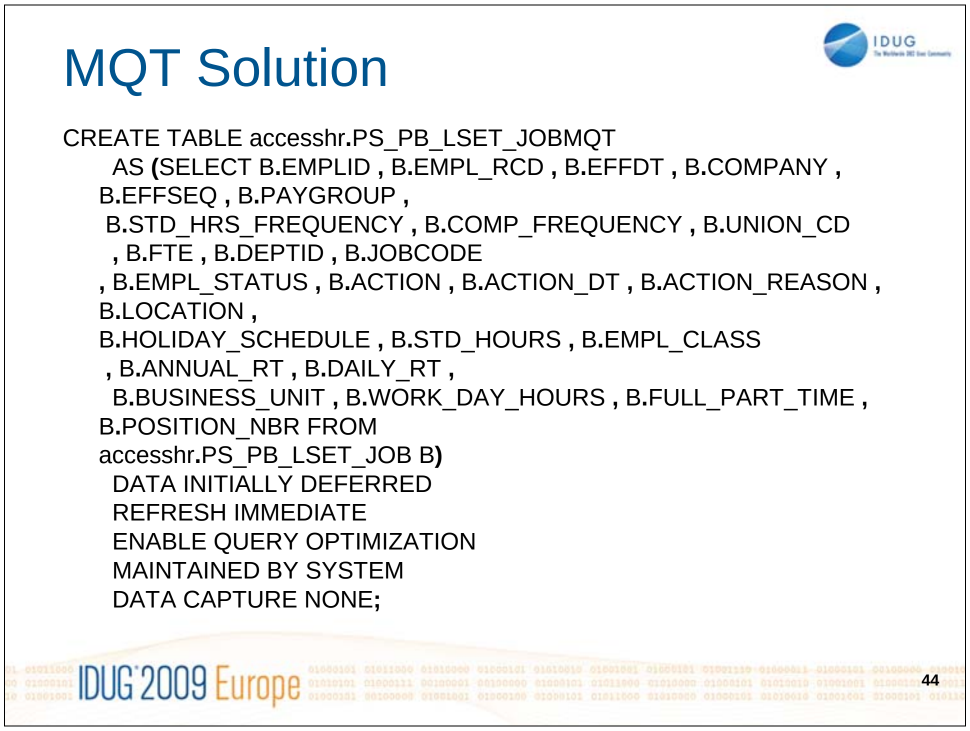


## Sub-optimal SQL MQT Candidate

```
SELECT B.EMPLID , B.EMPL_RCD , B.EFFDT , B.COMPANY ,
    B.EFFSEQ , B.PAYGROUP , B.STD_HRS_FREQUENCY ,
    B.COMP_FREQUENCY , B.UNION_CD , B.FTE , B.DEPTID ,
    B.JOBCODE, B.EMPL_STATUS , B.ACTION , B.ACTION_DT ,
    B.ACTION_REASON , B.LOCATION , B.HOLIDAY_SCHEDULE ,
    B.STD_HOURS , B.EMPL_CLASS , B.ANNUAL_RT ,
    B.DAILY_RT , B.BUSINESS_UNIT , B.WORK_DAY_HOURS ,
    B.FULL_PART_TIME , B.POSITION_NBR
  FROM PS_PB_LSET_JOB B
ORDER BY B.EMPLID , B.EMPL_RCD , B.EFFDT, B.EFFSEQ WITH
    UR
```

**What’s the obvious problem with this SQL?**

NOTES: This query was being run via PeopleSoft and could not be changed. Big problem was lack of a WHERE clause. An index could not be created for index only access due to number of columns in an index limitation in DB2 8.2. It was decided to consider an MQT as a solution.

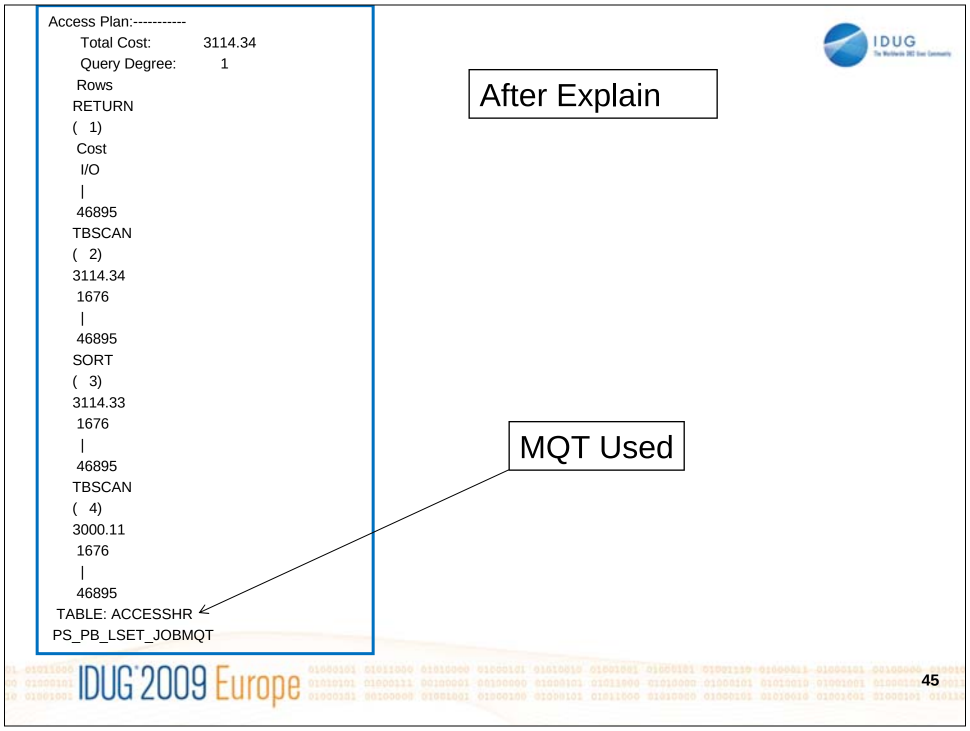


## MQT Solution

```
CREATE TABLE accesshr.PS_PB_LSET_JOBMQT
    AS (SELECT B.EMPLID , B.EMPL_RCD , B.EFFDT , B.COMPANY ,
  B.EFFSEQ , B.PAYGROUP ,
   B.STD_HRS_FREQUENCY , B.COMP_FREQUENCY , B.UNION_CD
    , B.FTE , B.DEPTID , B.JOBCODE
  , B.EMPL_STATUS , B.ACTION , B.ACTION_DT , B.ACTION_REASON ,
  B.LOCATION ,
  B.HOLIDAY_SCHEDULE , B.STD_HOURS , B.EMPL_CLASS
   , B.ANNUAL_RT , B.DAILY_RT ,
    B.BUSINESS_UNIT , B.WORK_DAY_HOURS , B.FULL_PART_TIME ,
  B.POSITION_NBR FROM
  accesshr.PS_PB_LSET_JOB B)
    DATA INITIALLY DEFERRED
    REFRESH IMMEDIATE
    ENABLE QUERY OPTIMIZATION
    MAINTAINED BY SYSTEM
    DATA CAPTURE NONE;
```

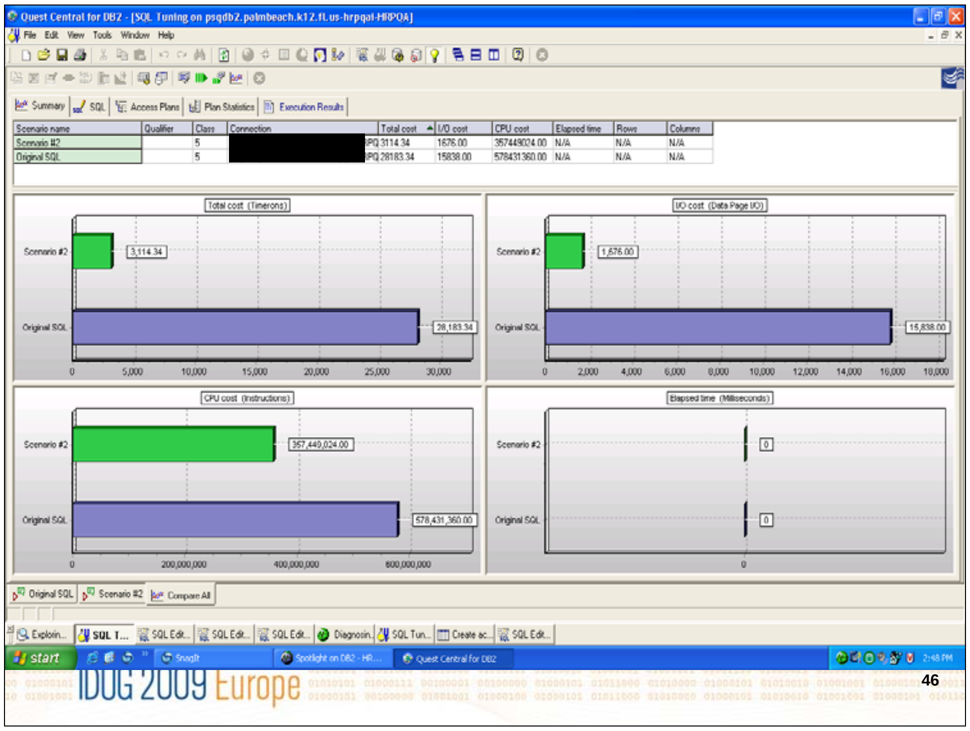
IDUG 2009 Europe 44

NOTES: Although not your typical MQT solution, the cost of accessing the MQT was significantly lower than doing a full table scan every time the statement was executed. Typically the statement was executed hundreds of thousands of time during each run. The cost of maintaining the MQT was considered but not found to be a problem as the table involved was updated from a batch job during off hrs and was not substantially affected.



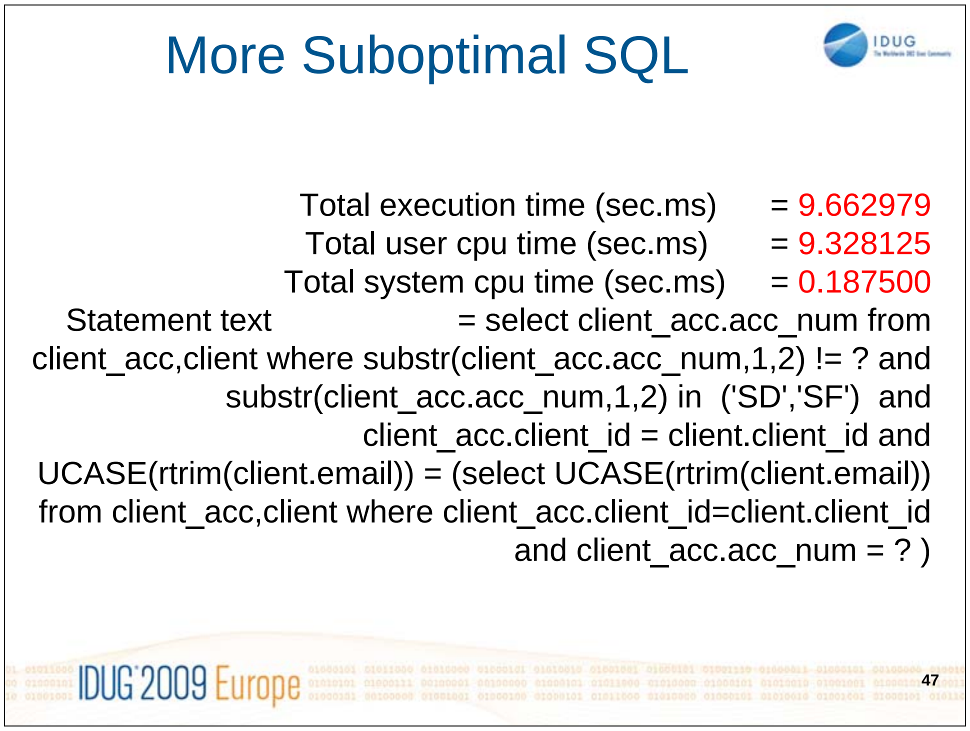
```
Access Plan:-----------
        Total Cost:            3114.34
        Query Degree:          1
       Rows
      RETURN
      (   1)
       Cost
        I/O
        |
       46895
      TBSCAN
      (   2)
      3114.34
       1676
        |
       46895
      SORT
      (   3)
      3114.33
       1676
        |
       46895
      TBSCAN
      (   4)
      3000.11
       1676
        |
       46895
  TABLE: ACCESSHR
 PS_PB_LSET_JOBMQT
```

NOTES: db2exfmt showing MQT being used. Timerons reduced from 28,000 to 3,114.



| Scenario name | Qualifier | Class | Connection | Total cost | I/O cost | CPU cost | Elapsed time | Rows | Columns |
|---|---|---|---|---|---|---|---|---|---|
| Scenario #2 | | 5 | [redacted] PQ | 3114.34 | 1676.00 | 357449024.00 | N/A | N/A | N/A |
| Original SQL | | 5 | [redacted] PQ | 28183.34 | 15838.00 | 578431360.00 | N/A | N/A | N/A |

NOTES: This slide shows a comparison of the costs of before and after creating the MQT. Total costs reduced from over 28,800 timerons down to 3,134.

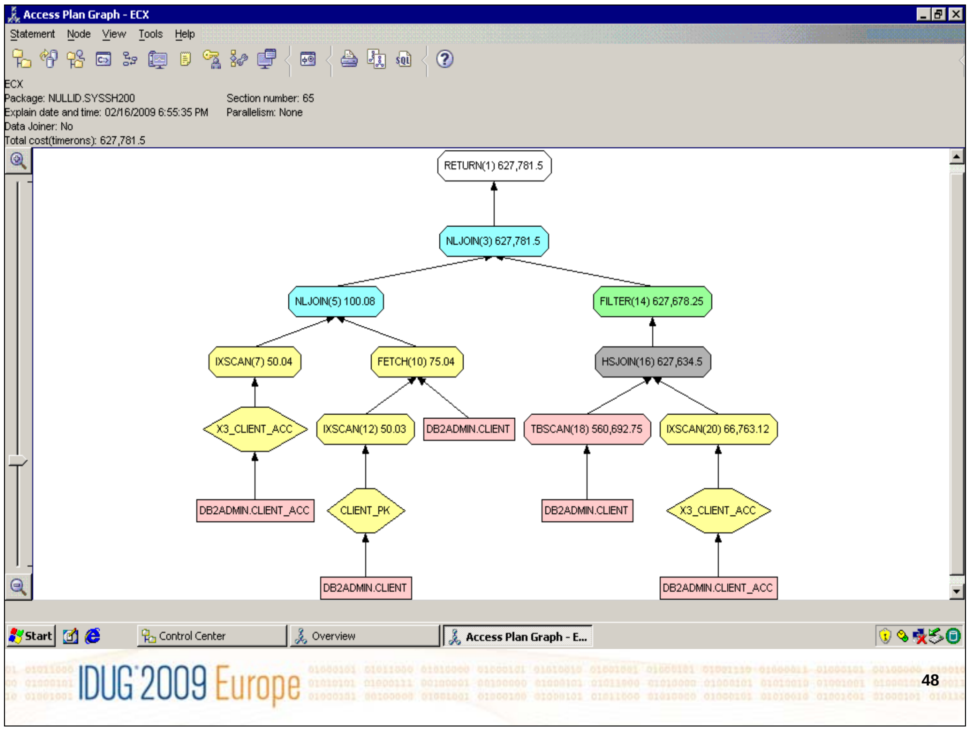


## More Suboptimal SQL

```
              Total execution time (sec.ms)     = 9.662979
              Total user cpu time (sec.ms)      = 9.328125
              Total system cpu time (sec.ms)    = 0.187500
Statement text                     = select client_acc.acc_num from
client_acc,client where substr(client_acc.acc_num,1,2) != ? and
substr(client_acc.acc_num,1,2) in  ('SD','SF')  and
client_acc.client_id = client.client_id and
UCASE(rtrim(client.email)) = (select UCASE(rtrim(client.email))
from client_acc,client where client_acc.client_id=client.client_id
and client_acc.acc_num = ? )
```

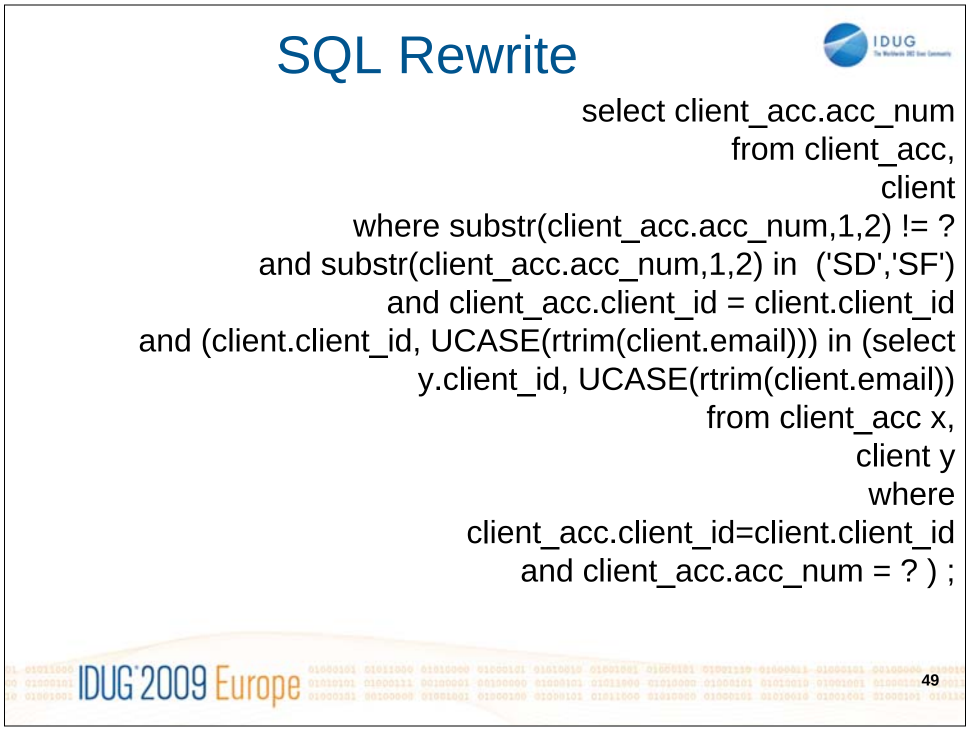
IDUG 2009 Europe 47

NOTES: This high cost statement was captured via a dynamic application snapshot and Top 10 SQL query. The SQL was then explained and analyzed.



ECX
Package: NULLID.SYSSH200 Section number: 65
Explain date and time: 02/16/2009 6:55:35 PM Parallelism: None
Data Joiner: No
Total cost(timerons): 627,781.5

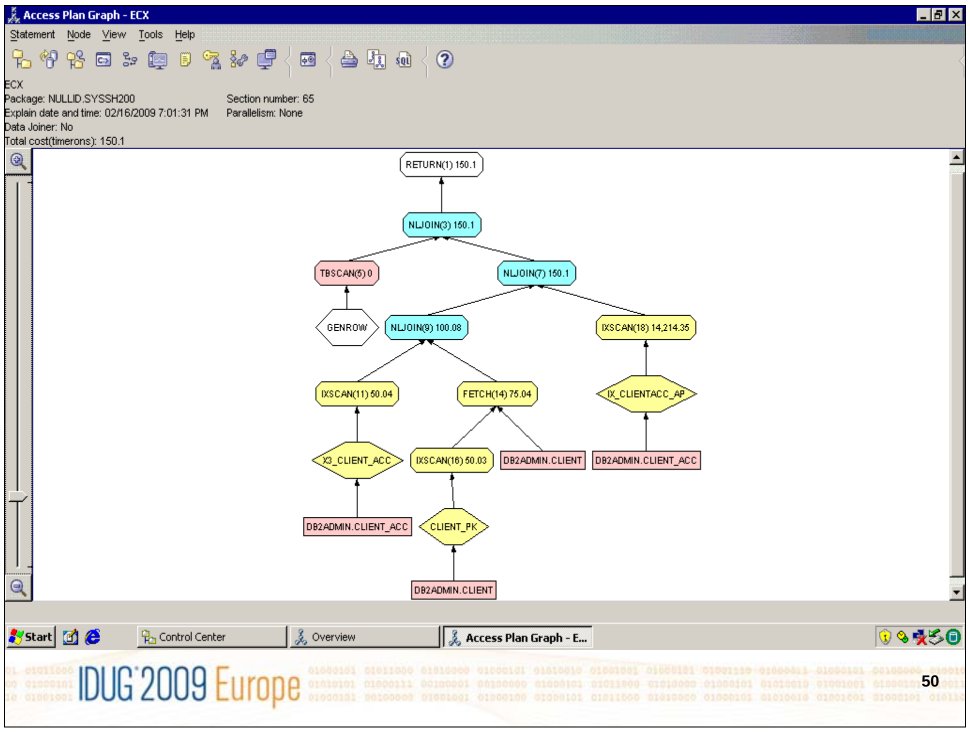
NOTES: Suboptimal high cost query of 627,781 timerons. This query was executed many times per second and was captured via a dynamic sql snapshot.



## SQL Rewrite

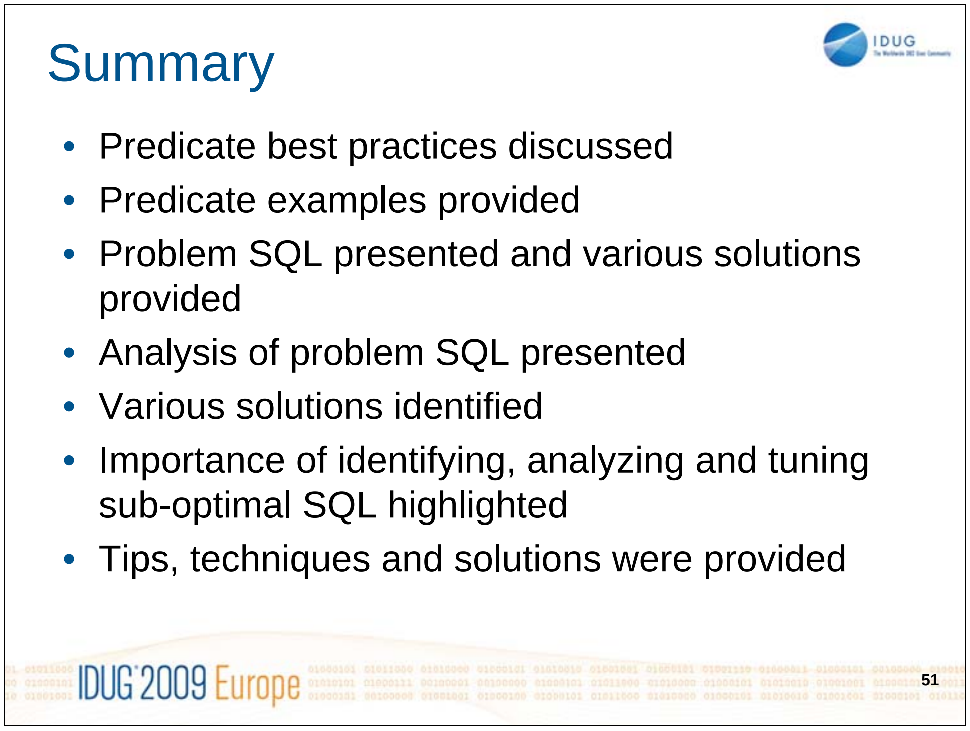
```
select client_acc.acc_num
from client_acc,
client
where substr(client_acc.acc_num,1,2) != ?
and substr(client_acc.acc_num,1,2) in  ('SD','SF')
and client_acc.client_id = client.client_id
and (client.client_id, UCASE(rtrim(client.email))) in (select
y.client_id, UCASE(rtrim(client.email))
from client_acc x,
client y
where
client_acc.client_id=client.client_id
and client_acc.acc_num = ? ) ;
```

NOTES: Rewritten, tuned SQL from previous slide. The query was rewritten to us IN instead of setting the whole query = to the subquery. The first method causes the optimizer to produce a plan that will read all the rows of the inner loop before it is sure there is only one match, whereas IN can cause the subquery to exit the inner loop using early out.



ECX
Package: NULLID.SYSSH200 Section number: 65
Explain date and time: 02/16/2009 7:01:31 PM Parallelism: None
Data Joiner: No
Total cost(timerons): 150.1

NOTES: The result of the rewrite is a much better plan and big reduction in timerons.



## Summary

- Predicate best practices discussed
- Predicate examples provided
- Problem SQL presented and various solutions provided
- Analysis of problem SQL presented
- Various solutions identified
- Importance of identifying, analyzing and tuning sub-optimal SQL highlighted
- Tips, techniques and solutions were provided

IDUG 2009 Europe 51

NOTES:

Session: C11

## SQL Tuning for Toughies, Top Tips, Techniques and Solutions

Phil Gunning

Gunning Technology Solutions, LLC

pgunning@gunningts.com

**THANK YOU!**